НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

До захисту допущено

Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)
"___" __червня__ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системне програмування»

спеціальності **123 «Комп'ютерна інженерія»**

на тему: «Система формування та контролю електронної черги для мобільних пристроїв»

Виконав: студент IV курсу, групи __КВ-63__
(шифр групи)

_____Кравчук Олександр Віталійович_____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник __старший викладач каф. СПіСКС Наливайчук М. В.__ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю, доц.каф.СПіСКС, к.т.н. Клятченко Я.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Системне програмування»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)
«___» ___червня___2020 р.

ЗАВДАННЯ
на дипломний проєкт студента
Кравчука Олександра Віталійовича

1. Тема проєкту «Система формування та контролю електронної черги для мобільних пристроїв», керівник проєкту Наливайчук Микола Васильович, старший викладач кафедри СПіСКС, кандидат технічних наук, затверджені наказом по університету від «25» травня 2020 р. №1181-С.

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту див. Технічне завдання.

4. Зміст пояснювальної записки: аналіз існуючих рішень та обґрунтування теми, аналіз розробки додатку на базі операційної системи Android та серверної частини, опис розробки та роботи кінцевого продукту і його тестування.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): структурна схема взаємодії модулів побудованої системи, алгоритм видалення користувача з черги, алгоритм створення черги, діаграма взаємодії класів клієнтського додатку, презентація.

6. Консультанти розділів проєкту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М., доц. каф. СПіСКС, к.т.н.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Видача завдання на дипломне проєктування.	12.11.2019	
2	Вивчення літератури за тематикою роботи.	24.11.2019	
3	Розроблення та узгодження технічного завдання.	19.12.2019	
4	Розроблення структури додатку.	27.01.2020	
5	Розроблення дизайну та графічних елементів.	11.02.2020	
6	Програмна реалізація додатку.	20.03.2020	
7	Тестування додатку.	26.04.2020	
8	Підготовка матеріалів текстової частини проєкту.	24.04.2020	
9	Підготовка матеріалів графічної частини проєкту.	07.05.2020	
10	Попередній захист дипломного проєкту.	20.05.2020	

Студент _____ Олександр КРАВЧУК
(підпис) (Ім'я та ПРІЗВИЩЕ)

Керівник проєкту _____ Микола НАЛИВАЙЧУК
(підпис) (Ім'я та ПРІЗВИЩЕ)

* Консультантом не може бути зазначено керівника дипломного проєкту.

АННОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (68 стор., 38 рис.).

В бакалаврському проєкті реалізовано систему формування та контролю електронної черги для мобільних пристроїв у вигляді клієнт-серверного додатку.

Метою проєкту є оптимізація організації різноманітних черг у повсякденному житті, а також підвищення комфорту її учасників, шляхом забезпечення інструментарієм, котрий був би простим, доступним та ефективним у використанні.

В даному проєкті було розроблено наступні компоненти:

- клієнт, у вигляді додатку на базі операційної системи Android, написаний за допомогою мови Java. Він надає інтуїтивно-зрозумілий користувацький інтерфейс, що дозволяє проводити такі маніпуляції з чергої як створення, керування та видалення;
- сервер на базі бібліотеки Flask, що містить у собі програмну логіку додатку та виконує обмін даними з користувачами за допомогою REST підходу до архітектури мережевих протоколів;
- база даних PostgreSQL, що зберігає дані у процесі функціонування додатку.

В результаті розробки було отримано кінцевий продукт, який має мінімальні вимоги для користувача (а саме: мобільний пристрій на базі Android та наявність підключення до мережі Internet) та, в свою чергу, надає потужний функціонал для зручного регулювання черг, що значно полегшує виконання даної тривіальної задачі.

Ключові слова: клієнт-серверний додаток, операційна система, Android, Java, Flask, REST підхід, база даних, PostreSQL, Internet.

ANNOTATION

Qualification work includes an explanatory note (68 p., 38 fig.).

The bachelor's project implements a system of creating and controlling the electronic queue for mobile devices in the form of a client-server application.

The aim of the project is to optimize the organization of various queues in everyday life, as well as increase the comfort of its participants by providing tools that would be simple, accessible and effective to use.

The following components have been developed in this project:

- client, in the form of an application based on the Android operating system, written in Java. It provides an intuitive user interface that allows to perform such manipulations as creating, managing and deleting queue;

- a server based on the Flask library, which contains logic of the application and exchanges data with users using the REST approach to the architecture of network protocols;

- PostgreSQL database, which stores data during the interacting with the application.

As the result of development was obtained a final product that has got minimal user requirements (namely, an Android-based mobile device and an Internet connection) and, in turn, provides powerful functionality for convenient managing of queues, that greatly facilitates the implementation of this trivial task.

Keywords: client-server application, operating system, Android, Java, Flask, REST approach, database, PostreSQL, Internet.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
1	А4	ІАЛЦ.045490.002 ТЗ	Система формування та	4		
			контролю електронної			
			черги для мобільних			
			пристроїв.			
			Технічне завдання			
2	А4	ІАЛЦ.045490.003 ТП	Система формування та	2		
			контролю електронної			
			черги для мобільних			
			пристроїв.			
			Відомість технічного			
			проєкту			
3	А4	ІАЛЦ.045490.004 ПЗ	Система формування та	68		
			контролю електронної			
			черги для мобільних			
			пристроїв.			
			Пояснювальна записка			
4	А4	ІАЛЦ.045490.005 Д1	Алгоритм видалення	1		
			користувача з черги.			
			Схема алгоритму			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045490.001 ОА		
Розробив		Кравчук О.В.			Система формування та контролю електронної черги для мобільних пристроїв **Опис альбому**	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Наливайчук М.В.					
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ КВ-63	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Романкевич В.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
5	А4	ІАЛЦ.045490.006 Д2	Взаємодія модулів системи	1		
			керування електронної			
			черги для мобільних			
			пристроїв.			
			Схема структурна			
6	А4	ІАЛЦ.045490.007 Д3	Алгоритм створення	1		
			черги.			
			Схема алгоритму			
7	А4	ІАЛЦ.045490.008 Д4	Діаграма взаємодії класів	1		
			клієнтського додатку.			
			Схема структурна			
8		Диск CD-ROM	Текст пояснювальної	1		
			записки. Текст програми.			
			Графічний матеріал			

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Система формування та контролю електронної черги для мобільних пристроїв».

Галузь застосування: використання в повсякденному житті для створення та регулювання електронних черг для мобільних пристроїв.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проєкту є створення системи для формування, керування та моніторингу електронних черг для мобільних пристроїв. Призначення роботи полягає у розробці кінцевого продукту та покращенні якості повсякденного життя.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації в періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

1.1 Вимоги до програмного продукту, що розробляється

- сумісність з операційною системою Android;
- наявність серверної частини, що реалізовує логіку додатку;
- можливість створення електронної черги;

- можливість приєднання користувачів до створених електронних черг за допомогою мобільного пристрою;
- функціонал для керування електронними чергами;
- інтуїтивно зрозумілий інтерфейс;

5.2 Вимоги до програмного та апаратного забезпечення користувача

- операційна система Android не нижче 4.4 версії;
- наявність доступу до мережі Wi-Fi (IEEE 802.11 b/g/n) або 3G/4G.

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Видача завдання на дипломне проектування	12.11.2019
2.	Вивчення літератури за тематикою роботи	24.11.2019
3.	Розроблення та узгодження технічного завдання	19.12.2019
4.	Розроблення структури додатку	27.01.2020
5.	Розроблення дизайну та графічних елементів	11.02.2020
6.	Програмна реалізація додатку	20.03.2020
7.	Тестування додатку	26.04.2020
8.	Підготовка матеріалів текстової частини проекту	24.04.2020
9.	Підготовка матеріалів графічної частини проекту	07.05.2020
10.	Оформлення технічної документації проекту	11.05.2020

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.004 ПЗ	Система формування та	68		
			контролю електронної			
			черги для мобільних			
			пристроїв.			
			Пояснювальна записка			
	А4	ІАЛЦ.045490.005 Д1	Алгоритм видалення	1		
			користувача з черги.			
			Схема алгоритму			
	А4	ІАЛЦ.045490.006 Д2	Взаємодія модулів системи	1		
			керування електронної			
			черги для мобільних			
			пристроїв.			
			Схема структурна			
	А4	ІАЛЦ.045490.007 Д3	Алгоритм створення	1		
			черги.			
			Схема алгоритму			
	А4	ІАЛЦ.045490.008 Д4	Діаграма взаємодії класів	1		
			клієнтського додатку.			
			Схема структурна			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045490.003 ТП		
Розробив		Кравчук О.В.			Система формування та контролю електронної черги для мобільних пристроїв **Відомість технічного проєкту**	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Наливайчук М.В				КПІ ім. Ігоря Сікорського, ФПМ КВ-63	
Консульт.							
Н. контроль		Клятченко Я.М.					
Зав. каф.		Романкевич В.О					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
		Диск CD-ROM	Текст ПЗ. Текст програми.	1		
			Графічний матеріал			

ЗМІСТ

ДОДАТКИ

Додаток А. Копії графічних матеріалів.

- ІАЛЦ.045490.005 Д1. Алгоритм видалення користувача з черги. Схема алгоритму.
- ІАЛЦ. 045490.006 Д2. Взаємодія модулів системи керування електронної черги для мобільних пристроїв. Схема структурна.
- ІАЛЦ. 045490.007 Д3. Алгоритм створення черги. Схема алгоритму.
- ІАЛЦ. 045490.008 Д4. Діаграма взаємодії класів клієнтського додатку. Структурна схема.

Додаток Б. Фрагменти програмного коду.

Додаток В. Презентація бакалаврського проєкту.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ОС – операційна система;

API – Application Programming Interface – програмний інтерфейс додатку;

JSON – JavaScript Object Notation – текстовий формат обміну даними;

REST – Representational State Transfer – архітектурний стиль взаємодії компонентів розподілений елементів додатку в мережі;

SDK – Software Development Kit – набір засобів розробки;

XML – Extensible Markup Language – стандарт побудови мов розмітки ієрархічно структурованих даних;

ВСТУП

Важко уявити сучасний світ без техніки. Вона оточує нас всюди – у побуті, навчальних закладах, на місцях роботи та відпочинку. Все це стало можливим за рахунок жаги людства до розвитку. У будь-який період часу в історії наші предки робили нові відкриття. Починаючи з винайдення простих інструментів для землеробства та закінчуючи біологічним 3D-принтером – все це так чи інакше змінювали певну частину людського життя.

Що ж спонукало людей завжди рухатися вперед? Однією з найвагоміших причин є бажання комфорту. Це відноситься не лише до умов проживання, таких як тепла домівка чи м’яке крісло, але й до зменшення зусиль, які людина прикладає для вирішення поставленої перед нею задачі. Саме спрощення виконання дій різної складності та характеру змушує людей розвиватися.

Поява перших комп’ютерів та всесвітньої мережі Інтернет стала наслідком даного розвитку та розпочала глобальну цифрову революцію. Внаслідок цього було запущено процес, що має назву оцифровування. Оцифровування – це зміни, що з’являються в усіх сферах суспільного життя та безпосередньо пов’язанні з використанням цифрових технологій. Прикладами цього можуть слугувати численні випадки переходу від зберігання інформації в паперових документах та ручних маніпуляцій з ними до спеціально розроблених програмних та апаратних засобів, що виконують дану роботу в рази ефективніше та є більш зручними у використанні.

Дана робота спрямована на продовження тенденції цифрової революції та вирішення такої проблеми як організація та керування чергами у повсякденному житті, шляхом розробки відповідного програмного забезпечення, що було б максимально зручним у використанні та доступним якнайширшому колу користувачів.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ

1.1 Поняття та види систем формування та контролю електронних черг

Система управління чергами - це сукупність різних модулів, призначених для ефективного керування чергам та запитами клієнтів. Процес формування черги та її поширення залежить від типу черги та її реалізації та визначається за допомогою теорії черг.

Теорія черг - це математичне вивчення ліній очікування (самих черг). Модель черги побудована так, щоб можна було передбачити тривалість черги та час очікування. Теорія черг, як правило, вважається галуззю операційних досліджень, оскільки результати часто використовуються при прийнятті бізнес-рішень щодо ресурсів, необхідних для надання послуги.

За типом розрізняють наступні черги:

- Структуровані черги. В даному випадку люди формують чергу у підготовленому для цього місці. Наприклад, у касах супермаркету, місцях роздрібної торгівлі, банках чи аеропортах. Дуже часто встановлюються системи управління розповсюдження квитків для черги (з номером або без нього), що надає менеджерам можливість ідентифікувати клієнта, а йому, в свою чергу, можливість безтурботного очікування. Розширюючи функціональність даної системи, можна також включити плановий прийом за попередньою домовленістю та дистанційне сповіщення через push-повідомлення.
- Неструктуровані черги. Для даного типу є характерним утворення черг людьми в різних, непередбачуваних місцях. Це часто трапляється в деяких формах роздрібної торгівлі, чергах на таксі, банкоматах та в періоди високого попиту у багатьох інших ситуаціях. У найбільш заповнених місцях встановлюються фізичні бар'єри та путівники для того, щоб формувати людей у лінію, коли вони прибувають.
- Черги на основі терміналів. Черги на основі терміналів часто використовуються у медичних, банківських, телекомунікаційних сферах

людської діяльності та у багатьох державних установах. Коли люди прибувають на місце, вони підходять до спеціального терміналу, обирають потрібну послугу та вводять основну інформацію про себе. Інформація організована в спеціальні форми та надається представнику служби обслуговування для забезпечення швидшого реагування на прибуття клієнтів. Системи, що базуються на терміналах, також включають систему відстеження інформації для бізнесу, з метою звітування про таку статистику як час очікування, обсяг трафіку та продуктивність персоналу.

- Черги на основі мобільних пристроїв. Система черг на основі мобільних пристроїв дозволяє клієнтам використовувати свій мобільний телефон для взаємодії з чергою та перегляду даних про неї в режимі реального часу. Клієнти можуть продовжувати свій день і не повинні чекати в зоні очікування. Коли клієнт є першим на черзі він отримує сповіщення через SMS або через push-повідомлення, а менеджер черги викликає його для обслуговування. Перш ніж проводити будь-які маніпуляції з чергою даний тип черги вимагає від клієнта встановлення відповідного програмного забезпечення на його мобільний пристрій.

Як окремі модулі, в системах управління чергою існують окремі засоби для вимірювання ключових параметрів. Вони розроблені для того, щоб допомогти менеджерам черг двома способами:

- за рахунок покращення обслуговування клієнтів;
- шляхом підвищення ефективності та зниження витрат;

Дані засоби використовують датчики підрахунку людей на входах та над чергами, щоб точно визначати їх кількість. У системах керування чергами на базі мобільних пристроїв підрахунок користувачів відбувається програмно, за допомогою підрахунку кількості людей, котрі підключилися / були обслуговані у зазначений період. Вбудовані алгоритми прогнозування можуть заздалегідь повідомити приблизну кількість замовлень, що будуть зроблені або кількість пунктів обслуговування, якої буде потрібно для задоволення попиту.

Інформаційні панелі з вище перерахованими даними доступні на моніторі комп’ютера або мобільного пристрою. Вони часто використовуються для надання різноманітної інформації, наприклад, даних про динамічну тривалість черги, або даних про час очікування та продуктивність оформлення замовлення на касі магазину. У випадку, коли продуктивність обслуговування знижується до мінімального рівня, наприклад, в супермаркетах чи банках, менеджери управління автоматично отримують відповідне повідомлення. Це надає їм можливість активного втручання для запобігання критичних ситуацій, що значно підвищує ефективність їхньої роботи.

Ключовими параметрами, що вимірюються у черзі, є:

- Кількість людей, що відвідало приміщення / чергу;
- Середня довжина;
- Середній час очікування;
- Загальний час очікування;

1.2 Існуючі аналоги систем керування чергами

Інтернет сервіс Crowd Control Direct на однойменному сайті пропонує власну систему контролю та керування електронною чергою за допомогою додаткового обладнання – а саме: бездротових пристроїв керування, спеціального екрану для відображення інформації та аудіо системи для сповіщення клієнтів. Ідея даної системи наступна: існує одна або кілька так званих сервісних позицій, які по можливості приймають наступного клієнта. Уповноважений можливістю керуванням черги на сервісні позиції, за допомогою натискання кнопки на бездротовому дистанційному пристрої, повідомляє, що його позиція вільна для прийняття наступного клієнта. На екран виводиться номер даної позиції, а також лунає аудіо сигнал, повідомляючи про це клієнту в голові черги (рисунок. 1.1). Крім цього, кілька таких систем можуть бути налаштовані на спільну роботу, дозволяючи викликати клієнтів з інших службових черг, що, наприклад, знаходяться в іншому приміщенні.

Рисунок 1.1 – Приклад системи формування та контролю чергою від Crowd Control Direct

З переліку компаній, що надають готові рішення у сфері систем управління чергою найбільшими є Wavetec, AurioPro та Qmatic. Всі вони пропонують здебільшого однаковий за своїм функціоналом продукт, що відрізняється лише вартістю та компонуванням пакетів послуг і деякими особливостями реалізації.

Wavetec має 2 основні версії системи формування та контролю чергою – лінійну та покращену. Лінійна версія є аналогічною до попередньо розглянутої від сервісу Crowd Control Direct. Покращена ж система містить у собі повний набір модулів, які пропонує компанія для організації системи керування чергою. Дана версія задовольняє різноманітні потреби масового обслуговування за допомогою багатогалузевих, багаторегіональних корпоративних рішень. Дана система дозволяє клієнтам та відвідувачам вставати у чергу беручи квиток різними способами, такими як термінал самообслуговування, веб-квиток чи мобільний додаток (рисунок 1.2).

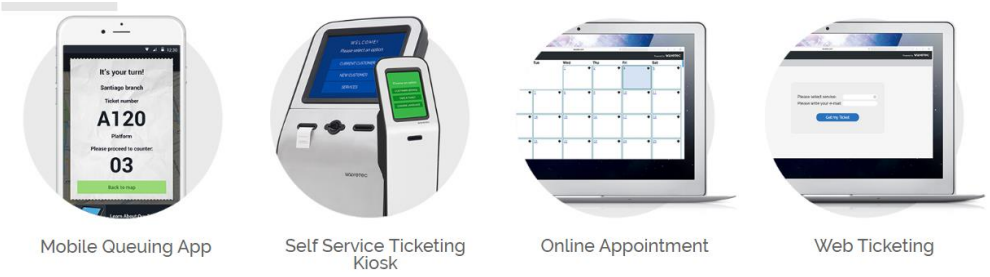

Рисунок 1.2 – Варіанти доступу до черги від Wavetec

AurioPro пропонує комплексне рішення питання черги у вигляді їхньої системи OptiQ. Це вдосконалена система масового обслуговування, що допомагає організувати та підтримувати повну карту переміщень клієнта (рисунок 1.3).

Рисунок 1.3 – Система OptiQ від компанії AurioPro

Існує 6 основних етапів проходження черги, які підтримує система OptiQ:

1. Генерація квитків.
2. Зона очікування.
3. Швидке сервісне направлення.
4. Обслуговування клієнтів.
5. Зворотній зв'язок.
6. Звіти та інформаційні панелі.

Компанія Qmatic є розробником системи Orchestra 7 (рисунок 1.4). Це модульна, масштабована платформа для управління потоком клієнтів. Основною її особливістю є те, що відштовхуючись від розміру середовища та кількості клієнтів можна обрати модулі, які є достатніми для даних умов, не купуючи при цьому весь пакет послуг. У разі потреби компанія гарантує безпроблемну інтеграцію розширень у існуючу систему та забезпечує потужний бізнес-аналіз.

Рисунок 1.4 – Продукт компанії Qmatic – Orchestra 7

Всі вище перераховані компанії є іноземними та не мають поширення на території СНГ. Це пов’язано зі складністю підтримки даної продукції у зв’язку з відстанню та низькою готовністю східноєвропейського ринку переходити на дані дорогі рішення.

Щодо аналогічних пропозицій на вітчизняному ринку, існує декілька фірм, що займаються організацією систем управління чергою (QMotion Suite, Human Queue Pro, Printec) однак всі вони також надають лише комплексні рішення даної задачі, орієнтовані на великі компанії-споживачі. До того ж, вони пропонують системи управління чергами базовані на терміналах, з відсутністю підтримки мобільних пристроїв.

1.3 Обґрунтування теми дипломного проєкту

Темою дипломного проєкту є створення системи формування та керування чергою для мобільних пристроїв. Як випливає з назви, розроблена система управління чергою повинна мати тип, що базується на мобільних пристроях.

Вибір даної теми ґрунтується на наступних трьох причинах:

1. Наявність даної проблеми у повсякденному житті. Проблема черг є неймовірно болючою в нашій країні та надокучила багатьом, так як вона зустрічається всюди. На щастя , не у всіх місцях вона є критичною. Наприклад, у супермаркетах чи місцях дрібної торгівлі завжди йде активний потік покупців і мало хто затримується на касі більше ніж на 2 хвилини. В даному випадку все проходить безболісно для клієнтів – ніякої штовханини, тисняви чи довгого очікування.

Однак, така картина є не всюди. Прийоми до чиновників у державних установах, до лікарів у поліклініках, черги в перукарнях та інші приклади, що спостерігаються у повсякденному житті, доводять це. Дана проблема існує навіть в рамках інститутів при здачі лабораторних робіт. У всіх вище перерахованих випадках система керування чергами для мобільних пристроїв значно полегшила б життя для обох сторін проблеми – для тих, хто приймає чергу, та для тих, хто в ній стоїть.

2. Потенційна користь та ефективність. Вирішення проблеми, описаної в попередньому пункті, мало б значний позитивний вплив на даний компонент життя. Повністю позбутися черг неможливо, однак система, описана в темі, покращила б перебування в них за рахунок наступних переваг:

- можливість ставати в чергу до приходу на позицію обслуговування;
- можливість відлучатися від черги без страху, так як при наближенні до її початку користувач отримує відповідне сповіщення;
- здатність завжди бачити стан черги;
- зручний контроль чергою для її власника;

Крім того, система управління чергою на основі мобільних пристроїв має низький поріг входження для користувачів у сучасних реаліях, так як на даний момент практично кожна людина має хоча б один мобільний пристрій та підключення до Інтернету.

3. Відсутність доступних аналогів. Ознайомившись з підрозділом 1.2 можна прослідкувати певну тенденцію, котра спостерігається у відсутності доступних аналогів даної системи.

Системи керування електронною чергою, розроблені іноземними компаніями є передовими в даній сфері та мають весь потрібний функціонал. Однак, разом з тим, вони мають і свої недоліки. Перш за все, це розміри системи та складність її встановлення і підтримки. Запропоновані аналоги складаються з багатьох модулів та потребують константного місцезнаходження. Тобто, система повинна бути встановлена у заздалегідь підготовленому для цього місці та зміна розташування є надзвичайно складною. По-друге, придбання повної системи не завжди є доцільним, зважаючи на різни запити користувачів. Дана проблема може бути вирішена модульними аналогами, таким як продукт компанії Qmatic – Orchestra 7. Однак, таке рішення є дорогим та складним у підтримці, що є значною перепоною у його використанні.

Так як серед вітчизняних компаній теж немає таких, котрі б пропонували альтернативні рішення, створення власної, доступної системи формування та керування чергою є перспективною задачею зі значним потенціалом в плані покращення життєвих умов.

Крім того, починаючи з 2019 року Україна взяла курс на оцифровування суспільного життя. Прикладом цього є мобільний додаток «Дія», що продовжує розроблятися як доступний сервіс для отримання державних послуг онлайн. Система формування та керування електронної черги для мобільних пристроїв стала б гідним наслідником даної справи.

1.4 Вибір реалізації системи керування чергою

Для створення системи формування та контролю електронної черги для мобільних пристроїв найкращим вибором є створення мобільного додатку. Створення відповідного програмного забезпечення для комп'ютеру є недоцільним, так як це значно зменшить доступність системи в побуті. Користувачу буде змушений постійно знаходитися за робочим місцем, що зменшує потенційну корить проєкту.

Формат мобільного додатку ідеально підходить для цілей даного проєкту, так як надає користувачам широкий набір можливостей у його використанні. Наявність додатку на мобільному пристрої дозволяє вільно пересуватися та проводити різні маніпуляції з чергою будь-де, де наявне підключення до мережі Інтернет. Крім того, мобільні додатки зазвичай мають набагато простішу структуру, ніж відповідне програмне забезпечення на комп'ютері та інтуїтивно зрозумілий інтерфейс.

Як мобільну платформу було обрано операційну систему Android з кількох причин (див. 2.1), так як вона задовольняє всі поставлені до неї вимоги та надає широкий набір інструментів розробки.

Крім додатку було вирішено реалізувати також серверну частину, що допомогла б користувачам взаємодіяти між собою шляхом обміну даних додатку з єдиним сервером. Дана можливість є життєво важливою, так як саме на основі взаємодії користувачів побудована суть системи.

Для збереження даних в процесі використання додатку було вирішено розділити дану задачу між власне сервером локально та базою даних, з тих причин, що деякі дані, у зв'язку з частим звертанням до них, краще мати у місці, доступ до якого здійснюється швидше.

Як підсумок, було обрано наступну реалізацію системи формування та керування чергою для мобільних пристроїв – це клієнт-серверний додаток, в якому роль клієнту відіграє мобільний додаток на базі операційної системи Android.

2. АНАЛІЗ МАТЕРІАЛІВ ДЛЯ РОЗРОБКИ ДОДАТКУ НА БАЗІ ОПЕРАЦІНОЇ СИСТЕМИ ANDROID

2.1 Обґрунтування вибору операційної системи Android

Перед початком розробки мобільного додатку потрібно обрати платформу (операційну систему), для якої він буде проєктуватися. Для цього потрібно розібратися з самим поняттям операційної системи, її існуючими реалізаціями для мобільних пристроїв та порівняти їх. Це і буде наведено нижче.

Операційною системою (ОС) називають програмне забезпечення, яке діє як інтерфейс між кінцевим користувачем та апаратним забезпеченням комп'ютера або мобільного пристрою. Іншими словами, ОС надає користувачу можливість керувати власним пристроєм, не вимагаючи при цьому поглиблених знань в його архітектурі. Наявність ОС є необхідною частиною для запуску інших додатків (програм), так як вона забезпечує середовище, в якому вони будуть працювати та виконувати свої завдання.

Серед операційних систем для мобільних пристроїв найпоширенішими є IOS, Android, Windows та Samsung. На рисунках 2.1-2.2 наведено відсоткове співвідношення між кількістю пристроїв на базі даних ОС (станом на квітень 2020) на ринку України та всього світу відповідно:

Рисунок 2.1 – Відсоткове співвідношення мобільних ОС на ринку України

Як можна побачити по статистиці, ОС Android опереджає найближчого свого конкурента більш ніж у 4 рази в межах України та у 2.5 рази по всьому світі.

Рисунок 2.1 – Відсоткове співвідношення мобільних ОС на всесвітньому ринку

З цього можна зробити висновок, що є сенс порівнювати ОС Android лише з його прямим конкурентом – IOS. Так як головною метою є розробка власного продукту, то і порівняння буде відбуватися по критеріях з даної сфери.

Якщо розглядати складність розробки, то розробка для IOS вважається порівняно легшою. Це, насамперед, пов'язано з мовою програмування Swift, що використовується при розробці для даної ОС. Порівняно з мовою програмування Java, на якій написано абсолютну більшість додатків на Android, Swift вимагає меншого об'єму коду для вирішення тих же задач, що зменшує час розробки програм.

В той же час, оскільки Android – це ОС з відкритим кодом (див. підрозділ 2.2), вона є більш гнучкою для розробки додатку, так як надає більш широкий спектр можливостей для створення власної унікальної функціональності з меншими обмеженнями.

Щодо інструментів та ціни розробки, то для Android існує інтегроване середовище розробки Android Studio, яке Google пропонує безкоштовно. Також розробники можуть безкоштовно розміщувати свої додатки в Google Play, хоча вони повинні сплатити одноразовий внесок у розмірі 25 доларів. У IOS офіційним інструментом розробки додатків є Xcode, котрий також доступний безкоштовно. Пакет для розробників Apple (компанії, що власне розробляє дану ОС) коштує 99 доларів на рік і включає можливість розміщення програм у iTunes App Store.

Підводячи підсумки, можна сказати, що обидві платформи надають користувачам можливість безкоштовної розробки додатків. Їх розміщення є

дорожчим у IOS, однак перевагою є більш простий процес розробки. Однак, ОС Android має 2 ключові переваги, що перехиляють вибір саме в її сторону:

1. Більша гнучкість розробки.
2. Значна перевага у поширеності мобільних пристроїв на локальному та всесвітньому ринку.

Останній компонент є критично важливим, так як додаток, що розробляється у даному проєкті, повинен бути доступним як можна більшій кількості користувачів.

Виходячи з усіх вище наведених аргументів, для розробки системи формування та керування електронної черги для мобільних пристроїв було обрано саме операційну систему Android.

2.2 Загальна інформація про операційну систему Android

При розробці якісного додатку важливо розуміти унікальні характеристики платформи, для якої він створюється. Для цього надалі буде розібрано загальні характеристики ОС Android та її особливості.

Android – це операційна система з відкритим кодом, побудована на модифікованій версії ядра Linux та призначена головним чином для мобільних пристроїв, таких як смартфони та планшети. Крім цього існує відповідний проєкт з відкритим кодом під керівництвом Google.

Користувальницький інтерфейс ОС заснований на прямому маніпулюванні, тобто він розроблений для пристроїв із сенсорним екраном, для реакції на прокручування, натискання, затискання та віртуальну клавіатуру. Дані функції виконуються під час використання внутрішніх апаратних засобів, таких як акселерометри, гіроскопи та датчики близькості. Хоча програмне забезпечення ОС Android в основному використовується для мобільних телефонів та планшетів, воно також може використовуватися для телевізорів та цифрових камер.

З переваг ОС Android можна виділити наступні:

- Відкритий код. Android був побудований на відкритому ядрі Linux таким чином, щоб розробники могли створювати власні мобільні додатки, які в повній мірі могли б користуватися всіма можливостями мобільного пристрою. Підтвердженням цього слугує те, що написана розробником програма може наприклад, залучати будь-яку з основних функцій телефону, таку як здійснення дзвінків, надсилання текстових повідомлень або використання камери, що дозволяє створювати більш зручні додатки для користувачів. Крім того, Android використовує власну віртуальну машину, розроблену для оптимізації пам'яті та апаратних ресурсів у мобільному середовищі, а відкритість коду дозволяє розширяти дану ОС шляхом додавання нових технології по мірі їх появи.
- Усі програми створюються рівноправними. Android не розрізняє основні програми телефону від сторонніх програм. Усі вони можуть бути побудовані з рівноправним доступом до можливостей телефону, що надає користувачам широкий спектр інструментів та послуг. Це надає можливість користувачам пристроїв, побудованих на платформі Android, повністю налаштувати телефон на свій смак. Розробники ж, в свою чергу, мають значно розширені можливості у створенні додатків для власних потреб.
- Швидка та проста розробка додатків. Android надає доступ до широкого вибору корисних бібліотек та інструментів, які можна використовувати для створення багатофункціональних додатків. Наприклад, розробник може створити додаток, який дозволяє користувачам переглядати місцезнаходження своїх друзів та отримувати сповіщення, коли вони знаходяться поблизу. Крім того, Android включає широкий набір інструментів, які були вбудовані на рівні платформи, що забезпечує розробникам високу продуктивність та глибоке розуміння створених додатків.

Android прагне бути найбільш безпечною та зручною операційною системою для мобільних платформ, замінюючи засоби безпеки операційної

системи для захисту даних програми та користувачів, системних ресурсів та забезпечення ізоляції програми від інших програм.

Для досягнення цих цілей дана ОС надає такі основні функції безпеки:

- Надійна безпека на рівні ОС через ядро Linux;
- Обов’язкове тестове середовище програми для всіх додатків;
- Безпечний міжпроцесорний зв’язок;
- Підписання програм сертифікатом Google Play;
- Дозволи, визначені додатком та надані користувачем;

2.3 Структура операційної системи Android

Операційна система Android складається з наступних компонентів (рисунок 2.1):

1. Linux Kernel, або Ядро Linux. Ядро - це серце операційної системи, яке керує запитами введення та виведення від програмного забезпечення. Ядро не взаємодіє безпосередньо з користувачем, а скоріше взаємодіє з оболонкою та іншими програмами, а також із пристроями апаратного обладнання в системі. Android використовує ядро Linux, яке включає в себе програми управління пам’яттю Android, налаштування безпеки, програмне забезпечення для управління енергією та драйвери апаратних засобів.

Драйвери - це програми, що керують апаратними пристроями. Наприклад, певний телефон на базі ОС Android має камеру. Ядро Android включає в себе драйвер камери, що дозволяє користувачеві надсилати команди до апаратної реалізації цієї ж камери.

Також, ядро, що використовується в даній ОС оснащене кількома спеціальними доповненнями, такими як Low Memory Killer (система управління пам'яттю, яка є більш строгою щодо збереження пам’яті), блокування пробудження (системна служба PowerManager), драйвер IPC Binder та іншими.

2. Hardware abstraction layer (HAL) / HAL interface definition language (HIDL), або апаратний рівень абстракції / мова опису інтерфейсу HAL відповідно. HAL – це специфічний апаратний рівень ОС Android, що дозволяє реалізувати функціональність, не впливаючи або не змінюючи систему вищого рівня. Реалізації HAL упаковані в модулі та завантажуються системою Android у відповідний час.

HIDL – це мова опису інтерфейсу для взаємодії між HAL та його користувачами, котра визначає структури даних та методів, що організовані в інтерфейси (подібні класу), які, в свою чергу, збираються в пакети. Також HIDL призначена для використання у міжпроцесорному зв'язку.

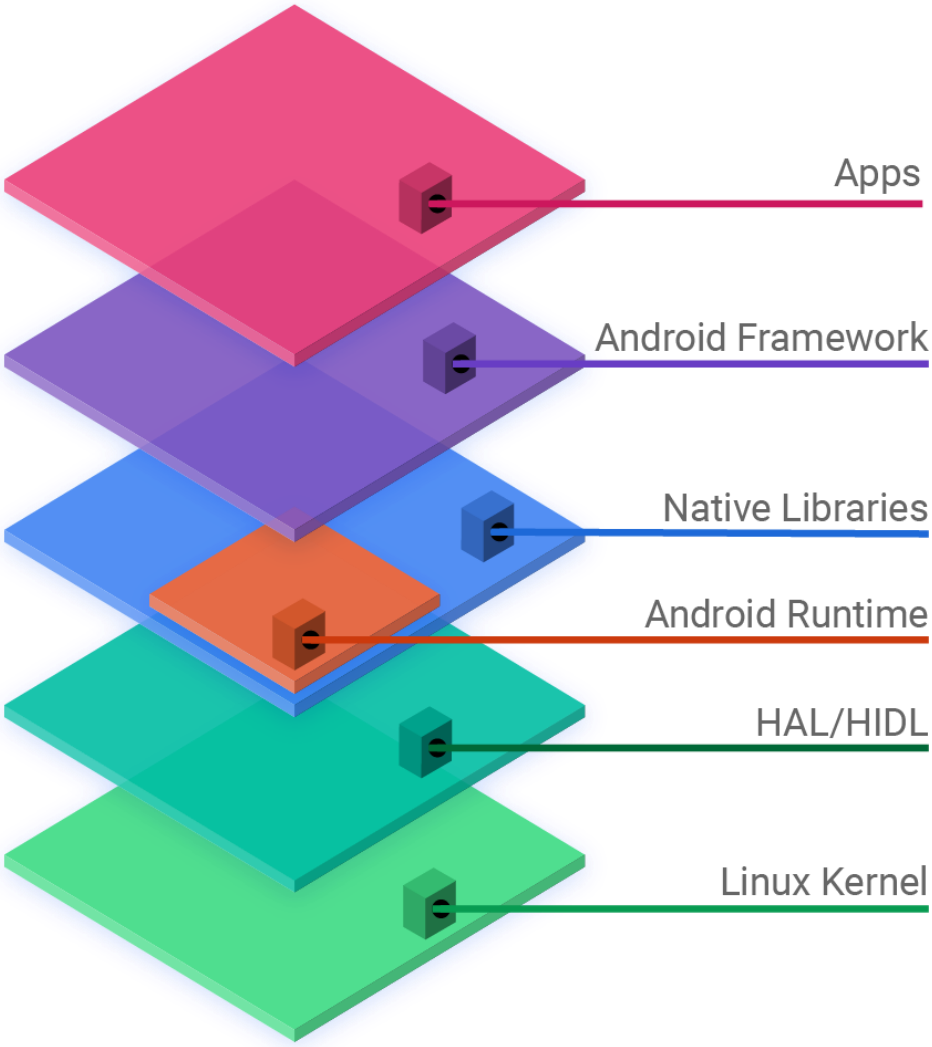

Рисунок 2.3 – Архітектура операційної системи Android

3. Native libraries, або стандартні бібліотеки. Даний рівень містить стандарні бібліотеки ОС Android. Всі вони написані на C/C++, але викликаються через Java-інтерфейси. Бібліотеки можна розглядати як набір інструкцій, які кажуть пристрою, як обробляти різні види даних. Наприклад, медіа бібліотека, що підтримує відтворення та запис різних форматів аудіо, відео та зображень. Також там знаходяться бібліотеки, які включають в себе роботу з 2D графікою, бібліотеки SSL (відповідають за безпеку Інтернету), бібліотеку тривимірного прискорення (для пристроїв з акселерометрами), базу даних SQL (SQLite) та стандартну бібліотеку веб-браузера (WebKit).

4. Android Runtime, або час виконання Android. Даний рівень включає в себе віртуальну машину Dalvik, що є своєрідною віртуальною машиною Java. Віртуальна машина - це програмне забезпечення, яке поводиться так, ніби це незалежний пристрій із власною операційною системою. Віртуальна машина може бути запущена на комп'ютері, який працює на зовсім іншій ОС, ніж ОС фізичної машини. Android використовує Dalvik для запуску кожної програми як власного процесу. Це важливо з кількох причин:

- жодна програма не залежить від іншої;
- якщо програма виходить з ладу, це не повинно впливати на інші програми, що працюють на пристрої;
- це спрощує управління пам’яттю;

Dalvik запускає файли розширення .dex, які приховані під час компіляції від стандартних файлів класу та jar. Файли .dex є більш компактними та ефективними, ніж файли класу, що має важливе значення для пристроїв з обмеженою пам’яттю та акумуляторним живленням. Основні бібліотеки Java також є частиною цього рівня. Вони написані на мові програмування Java, як і все вище цього рівня. Тут Android надає значну множину пакетів версії Java 5 Standard Edition, включаючи колекції, введення / виведення тощо.

5. Application framework, або рівень додатків. Сюди входять програми, які керують основними функціями телефону, такими як розподіл ресурсів, телефонні програми, перемикання між процесами чи програмами та

відстеження фізичного розташування телефону. Розробникам додатків дозволяється використовувати ці послуги у своїй програмі, через що даний рівень користується високою популярністю серед них.

6. На найвищому рівні ОС Android знаходяться додатки, що й формують користувальницький інтерфейс. Залежно від їх джерела, існують наступні дві групи:

- Попередньо встановлені програми. Android містить в собі набір попередньо встановлених програм, включаючи телефон, електронну пошту, календар, веб-браузер та контакти. Вони функціонують як користувацькі програми та забезпечують основні можливості пристрою. Попередньо встановлені програми можуть бути частиною платформи Android з відкритим кодом, або вони можуть бути розроблені компанією-продавцем для певного пристрою.
- Програми, встановлені користувачем. Android надає відкрите операційне середовищу, яке підтримує будь-яку сторонню програму, а Google Play пропонує користувачам сотні тисяч додатків. Крім того, це надає можливість користувачам розробляти, встановлювати та використовувати власні додатки.

2.4 Поняття рівня Android API

Ще однією важливою темою, без якої неможливо починати розгляд розробки Android додатку є Android API рівень. Також важливо розуміти поняття ідентифікатору рівня API та його роль у забезпеченні сумісності додатку з пристроями, на яких він може бути встановлений. Нижче наведено визначення даних понять.

Android API рівень – це просте ціле число, яке однозначно ідентифікує ревізію робочого оточення програмних бібліотек (framework API revision), яка надається версією операційної системи Android. Платформа Android надає

framework API, яке може бути використане додатками. Рівні API зазвичай визначають, який набір функцій доступний для розробника при даному значенні. Зі збільшенням рівня API доступна функціональність, як правило, збільшується (хоча деякі з функцій можуть бути застарілими).

Framework API складається з:

- базового набору пакетів і класів;
- набору елементів XML і атрибутів для декларування в файлі маніфесту (manifest file);
- набору елементів XML і атрибутів для декларування та доступу до ресурсів;
- набору намірів (Intents);
- набору дозволів доступу до ресурсів пристрою та системи, які додаток може запитати, а також захист обмеження доступу, що вбудований в систему.

Кожна наступна версія платформи Android може включати оновлення до програмного забезпечення API, яке вона постачає.

Оновлення для framework API розроблені таким чином, щоб новий API залишався сумісним з більш ранніми версіями API. Тобто, більшість змін в API є додатками та вводять нові функції або функції заміни. По мірі оновлення функціональності API застарілі частини, що були замінені, не видаляються. Таким чином існуючі програми все ще можуть їх використовувати. У дуже невеликій кількості випадків частини API можуть бути змінені або видалені, хоча зазвичай такі зміни потрібні лише для забезпечення надійності API та безпеки програми / системи. Усі інші частини API з попередніх версій передаються без змін.

API framework задається за допомогою цілогочисленого ідентифікатора, який називається "рівень API". Кожна версія платформи Android підтримує рівно один рівень API, хоча існує неявна підтримка для всіх попередніх рівнів API (до 1 рівня API). Початковий випуск платформи Android надавав API 1 рівня та наступні версії ОС поступово підвищували його.

Нижче (рисунок 2.3) вказано 7 перших та 7 останніх рівнів API, які підтримують відповідні версієї платформи Android.

Ідентифікатор рівня API виконує ключову роль у забезпеченні комфортної роботи розробників додатків:

- це дозволяє платформі Android описати максимальну версію API-версії, яку вона підтримує;
- це дозволяє програмам описувати рівень API, який вони потребують;
- це дозволяє системі узгоджувати встановлення програм на пристрої користувача, таким чином, що несумісні з версіями додатки не встановлюються.

Кожна версія платформи Android зберігає свій ідентифікатор рівня API всередині самої системи.

Програми можуть використовувати елемент маніфесту, наданий framework API – <uses-sdk> – для опису мінімального та максимального рівнів API, під яким вони можуть працювати, а також бажаного рівня API, який вони розроблені підтримувати.

Даний елемент маніфесту пропонує три ключові атрибути:

- android: minSdkVersion - Вказує мінімальний рівень API, на якому програма може працювати. Значення за замовчуванням - "1".
- android: targetSdkVersion - визначає рівень API, на якому програма призначена для роботи. У деяких випадках це дозволяє додатку використовувати елементи маніфесту або поведінку, визначені на цільовому рівні API, а не обмежуватись лише тим, що визначено для мінімального рівня API.
- android: maxSdkVersion - визначає максимальний рівень API, на якому програма може працювати.

Наприклад, щоб вказати мінімальний рівень API, який потрібен додатку для запуску, програма включить у свій маніфест елемент <uses-sdk> з

Platform Version	API Level	VERSION_CODE
Android 10.0	29	Q
Android 9	28	P
Android 8.1	27	O_MR1
Android 8.0	26	O
Android 7.1.1 Android 7.1	25	N_MR1
Android 7.0	24	N
Android 6.0	23	M
…	…	…
Android 2.1.x	7	ECLAIR_MR1
Android 2.0.1	6	ECLAIR_0_1
Android 2.0	5	ECLAIR
Android 1.6	4	DONUT
Android 1.5	3	CUPCAKE
Android 1.1	2	BASE_1_1
Android 1.0	1	BASE

Рисунок 2.3 – Таблиця відповідностей версій ОС Android до рівня API

атрибутом android: minSdkVersion. Значення android: minSdkVersion буде цілим числом, що відповідає рівню API найдавнішої версії платформи Android, під якою програма може працювати.

Коли користувач намагається встановити додаток або під час відновлення програми після оновлення системи, ОС Android спочатку перевіряє атрибути <uses-sdk> у маніфесті програми та порівнює значення зі своїм внутрішнім рівнем API.

Система дозволяє розпочати встановлення додатку лише за умови дотримання наступних умов:

– якщо оголошено атрибут Android: minSdkVersion, його значення має бути меншим або рівним цілому числу рівня API системи. Якщо це значення не оголошено, система передбачає, що програма вимагає API рівня 1.

– якщо оголошено атрибут Android: maxSdkVersion, його значення повинно бути рівним цілому цілому числу рівня API. Якщо це значення не оголошено, система передбачає, що програма не має максимального рівня API.

Оголошений в маніфесті програми, елемент <uses-sdk> може мати наступний вигляд (рисунок 2.4):

```
<manifest>
  <uses-sdk android:minSdkVersion="5" />
  ...
</manifest>
```

Рисунок 2.4 – Оголошення елементу <uses-sdk> у файлі маніфесту додатку

Основна причина, за якою програма оголошує рівень API в android: minSdkVersion, полягає в тому, щоб сказати системі Android, що вона використовує API, який було введено у визначеному рівні API. Якщо додаток потрібно було б якось встановити на платформі з нижчим рівнем API, то процес встановлення зазнав б аварії під час намагання отримання доступу до API, якого не існує. Система запобігає такому результату, не допускаючи встановлення додатку, якщо найнижчий рівень API, який він вимагає, вище, ніж у версії платформи на цільовому пристрої.

Наприклад, пакет android.appwidget був представлений у 3 рівні API. Якщо додаток використовує цей API, він повинен оголосити атрибут android: minSdkVersion зі значенням "3". Далі додаток буде встановлено на платформах, таких як Android 1.5 (API рівень 3) та Android 1.6 (API рівень 4), але не на платформах Android 1.1 (API рівень 2) та Android 1.0 (API рівень 1).

При розробці Android додатку необхідно також враховувати наступні моменти:

1. Пряма сумісність програми. Програми для Android, як правило, сумісні з новими версіями платформи, оскільки майже всі зміни в framework API направленні на додавання нової функціональності зі зберіганням старої. Однак, існують виключення, при яких програма використовує частину API, яка у більш пізніх версіях є видаленою. Розробник повинен слідкувати за прямою сумісністю програми та усувати проблеми у разі їх виникнення.

2. Зворотня сумісність програми. Програми для Android не обов'язково сумісні з більш ранніми версіями платформи Android. Хоча навряд чи пристрій, що працює на Android, буде переведено на попередню версію платформи, важливо усвідомити, що на ринку, ймовірно, буде багато пристроїв, які працюють із більш ранніми версіями платформи. Даний нюанс також повинен бути врахований.

3. Вибір версії платформи та рівня API. Перед розробкою додатку потрібно обрати версію платформи, для якої він призначений. Якщо створюється додаток, що використовує API або системні функції, представлені в останній версії платформи, то слід встановити атрибут android: minSdkVersion на рівні API останньої версії платформи.

2.5 Android SDK

Наступним кроком до розробки власного Android додатку є знайомство з Android SDK (Software Development Kit). Хоча існує багато різних мов програмування та безліч IDE (інтегрованих середовищ розробки), які можна використовувати для створення програми, SDK є завжди константною частиною розробки.

Android SDK (комплект для розробки програмного забезпечення) – це набір інструментів розробки, що використовуються для розробки програм для платформи Android. Незалежно від того, чи додаток створюється на мові Java,

Kotlin або C#, SDK є необхідною умовою для його запуску на пристрої Android та отримання доступ до унікальних функцій ОС.

Щоразу, коли Google випускає нову версію Android, також виходить відповідна версія SDK. Щоб мати змогу писати програми з найновішими функціями, розробник повинен завантажити та встановити SDK відповідної версії для конкретного телефону.

Платформи розробки, сумісні з SDK, включають такі операційні системи, як Windows (XP або новіші версії), Linux (будь-який останній дистрибутив Linux) та Mac OS X (10.4.9 або новішої версії). Компоненти Android SDK можна завантажити окремо.

Android SDK включає в себе наступне:

- обов'язкові бібліотеки;
- налагоджувач (debugger);
- емулятор;
- відповідна документація для програмних інтерфейсів Android (API);
- зразки сирців;
- документація для ОС Android;

Хоча SDK можна використовувати для написання програм Android у командному рядку, найпоширенішим методом є використання IDE.

2.6 Основи розробки додатку на базі ОС Android

Додатки для ОС Android можуть бути написані за допомогою мов Kotlin, Java та C++. Засоби Android SDK компілюють код разом із будь-якими додатковими файлами ресурсів у APK, що представляє собою архівний файл із суфіксом .apk.

APK (Android Package Kit) – це формат файлу пакунків, який використовується операційною системою Android для розповсюдження та встановлення мобільних додатків. Так само, як системи Windows (PC)

використовують файл .exe для встановлення програмного забезпечення, APK робить те ж саме для Android.

Кожен додаток для ОС Android живе у власній скриньці безпеки, захищеній такими способами як:

- операційна система Android - багатокористувацька система Linux, в якій кожен додаток - це інший користувач;
- за замовчуванням система призначає кожному додатку унікальний ідентифікатор користувача Linux (ідентифікатор використовується тільки системою і невідомий додатку). Система встановлює дозволи для всіх файлів програми, щоб доступ до них мав лише ідентифікатор користувача, призначений для цього додатка;
- кожен процес має власну віртуальну машину (VM), тому код програми працює ізольовано від інших додатків;
- за замовчуванням кожен додаток працює у власному Linux-процесі. Система Android запускає процес, коли будь-який з компонентів програми потрібно виконати, а потім вимикає процес, коли він більше не потрібен або коли система повинна відновити пам'ять для інших додатків.

Система Android реалізує принцип найменшої пільги. Тобто кожен додаток за замовчуванням має доступ лише до компонентів, які потрібні йому для виконання своєї роботи, і не більше того. Це створює безпечне середовище, в якому додаток не може отримати доступ до частин системи, для яких дозволу не надано. Однак існують наступні способи для обміну даними з іншими додатками та для доступу до системних служб:

1. Можна домовитись, щоб два додатки мали спільний доступ до одного ідентифікатора користувача Linux, і в цьому випадку вони мають доступ до файлів один одного.
2. Додаток може запитувати дозвіл на доступ до даних пристроїв, таких як розташування пристрою, камера та з'єднання Bluetooth. Користувач повинен явно надати ці дозволи.

Найважливішими складовими у розробці додатків для Android є їх додатків. Кожен компонент - це точка входу, через яку система або користувач можуть взаємодіяти з додатком. Деякі з компонентів можуть також залежати від інших.

Усього існує чотири різні типи компонентів додатку:

- Activities;
- Services;
- Broadcast receivers;
- Content providers;

Кожен тип служить окремій цілі та має чіткий життєвий цикл, який визначає як саме компонент створюється та знищується. Далі детальніше описано дані типи компонентів додатку:

1. Activities. Activities – це точка входу для взаємодії користувача з інтерфейсом додатку. Вони являють собою функціональність одного екрану з користувальницьким інтерфейсом. Наприклад, програма електронної пошти може мати одну Activity, яка показує список нових електронних листів, іншу Activity для створення електронної пошти та ще одну Activity для читання електронних листів. Незважаючи на те, що Activity працюють разом, щоб створити єдиний додаток електронної пошти, кожна з них не залежить від іншої. Таким чином, інша програма може розпочати будь-яку з них, якщо програма електронної пошти дозволяє це. При реалізації власних Activity потрібно створювати класи, які є підкласами вбудованого класу Activity. В додатку обов’язково повинна існувати принаймні одна Activity яка що має статус MainActivity.

2. Services (сервіси). Сервіси – фонові дії, які виконує додаток. Це можуть бути тривалі операції, такі як програвання музики під час серфінгу в Інтернеті. Для сервісів, можливо, знадобляться інші допоміжні компоненти типу Service для виконання певних завдань. Основна мета сервісів - забезпечити безперервну роботу програми, не порушуючи при цьому взаємодії з користувачем.

3. Broadcast receivers. Даний компонент використовується для відповіді на повідомлення від інших програм або системи. Наприклад, коли заряд акумулятора телефону низький, ОС Android видає широкомовне (Broadcast) повідомлення, що запускає функцію, котра сповіщує про це користувача та пропонує перейти до енергозберігаючого режиму., Після отримання даного повідомлення додаток вживає відповідних дій. Broadcast Receiver - це підклас класу BroadcastReceiver.

4. Content provider. Цей компонент керує набором спільних даних додатку та може зберігати їх у файловій системі, у базі даних SQLite, в Інтернеті чи будь-якому іншому постійному місці зберігання, до якого розроблюваний додаток може отримати доступ. За допомогою Content provider інші додатки можуть запитувати або змінювати дані, якщо додаток-власник надає такий дозвіл. Передачі даних від одного додатку до інших відбувається за запитом та обробляється класом ContentResolver. Даний клас реалізує API, який дозволяє іншим програмам здійснювати транзакції. Будь-який Content provider повинен реалізовувати батьківський клас ContentProvider.

Останнім важливим компонентом для розробки додатку на базі ОС Android є його структура (рисунок 2.5). Далі наведено опис виділених на даному рисунку елементів:

1. Папка manifest. Перш ніж операційна система Android може запустити компонент програми, вона повинна знати, що даний компонент існує. Виконується це за допомогою читання файлу маніфесту програми AndroidManifest.xml. Додаток повинен задекларувати всі його компоненти в цьому файлі.

Окрім декларації компонентів програми, маніфест додатково виконує ряд таких речей, як:

- визначення будь-яких дозволів користувачів, які вимагає додаток, (наприклад, доступ до Інтернету чи доступ для читання до контактів користувача);

– визначення мінімального рівня API, що необхідний додатку, на основі API, який даний додаток використовує;

– визначення апаратних та програмних функцій, що використовуються або вимагаються додатком (наприклад, камера, послуги Bluetooth або екран з декількома дотиками);

– визначення бібліотек API, з якими потрібно зв'язувати додаток, таких як бібліотека Google Maps, тощо.

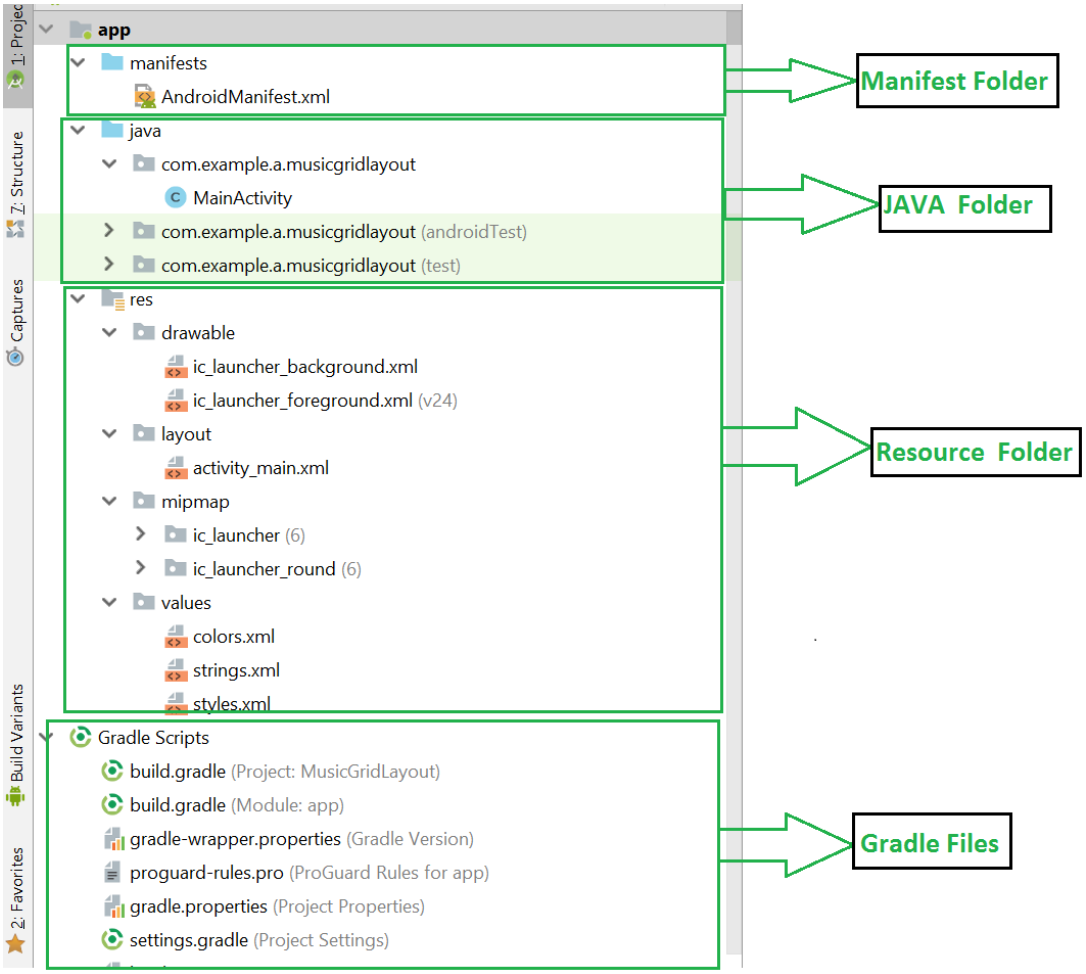

Рисунок 2.5 – Структура додатку, що розробляється на базі ОС Android

2. Папка Java. Складається з файлів java, необхідних для виконання фонового процесів програми. В них описується логіка роботи програми, а саме функціональності кнопок, різноманітні обчислення, спосіб зберігання даних, видачу тостів (невеликих спливаючих повідомлень), код функцій тощо. Кількість цих файлів залежить від розробника та створених Activities.

3. Папка res, або папка ресурсів. Дана папка містить у собі повний набір ресурсів, які використовуються у додатку. Вона складаються з під-папок, таких як drawable, layout, minmap та values.

Під-папка drawable зазвичай містить у собі зображення та описи елементів, що використовуються у додатку. Під-папка layout (макет) складається з файлів XML, що визначають розмітку інтерфейсу користувача. Під-папка minmap складається з файлів ресурсів, таких як аудіофайли. Доступ до них у програмі здійснюється через R.raw.filename. Під-папка values використовуються для зберігання сталих строкових значень, цілих чисел та кольорів, що часто використовуються у програмі (вважається безпечним зберігання даних значень саме у цей спосіб). У програмі вони також використовуються через відповідне посилання (R.color, наприклад), тому при зміні цих даних у цій папці вони відповідно змінюють скрізь у всій програмі.

4. Gradle. Gradle – це набір інструментів, який використовується для управління процесом компіляції програми, що дозволяє визначити конфігурації даного процесу. Кожна конфігурація збірки може мати власний набір коду та ресурсів, використовуючи при цьому деталі, загальні для всіх версій програми. Плагін Android для Gradle працює з набором інструментів збірки, щоб забезпечити процеси та налаштування, що характерні для створення та тестування додатків Android. Gradle працює незалежно від Android Studio. Гнучкість системи компіляції Android дозволяє використовувати власні конфігурації компіляції додатку без зміни основних вихідних файлів програми.

3. АНАЛІЗ МАТЕРІАЛІВ ДЛЯ РОЗРОБКИ СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ

3.1 Мережева модель «клієнт-сервер»

Для забезпечення обміну даними між користувачами було вирішено використати архітектуру типу «клієнт-сервер». Вона дозволяє користувачам взаємодіяти один з одним не зважаючи на відстань. Головною умовою є підключення до мережі Інтернет та встановлений додаток. Розглянемо детальніше дану модель.

Архітектура «клієнт – сервер» - це обчислювальна модель, в якій сервер розміщує, надає та керує більшістю ресурсів і послуг, які використовує клієнт. Цей тип архітектури має один або кілька клієнтських комп'ютерів або мобільних пристроїв, підключених до центрального сервера через мережеве або інтернет-з'єднання. Сервери чекають надходження запитів від клієнтів, а потім відповідають на них. В ідеалі, сервер забезпечує стандартизований інтерфейс для клієнтів, що знімає їх потреб в знаннях специфіки системи (тобто апаратного та програмного забезпечення), що вкористовується.

Ця обчислювальна модель особливо ефективна, коли клієнти та сервер мають різні завдання, які вони регулярно виконують. Наприклад, в обробці даних в лікарні на клієнтському комп'ютері може бути запущена програма для введення інформації про пацієнтів, поки серверний комп'ютер працює з іншою програмою, яка управляє базою даних, що відповідає за збереження інформації. Багато клієнтів можуть отримати доступ до серверу одночасно і, в той же час, клієнтський комп'ютер може виконувати інші завдання, наприклад, надсилання електронної пошти.

Архітектура «клієнт – сервер» також відома як мережева обчислювальна модель або мережа клієнт / сервер, оскільки всі запити та послуги постачаються по мережі. У своїй найпростішій формі Інтернет також базується на архітектурі «клієнт – сервер», де веб-сервери обслуговують одночасно багатьох користувачів даних веб-сайтів.

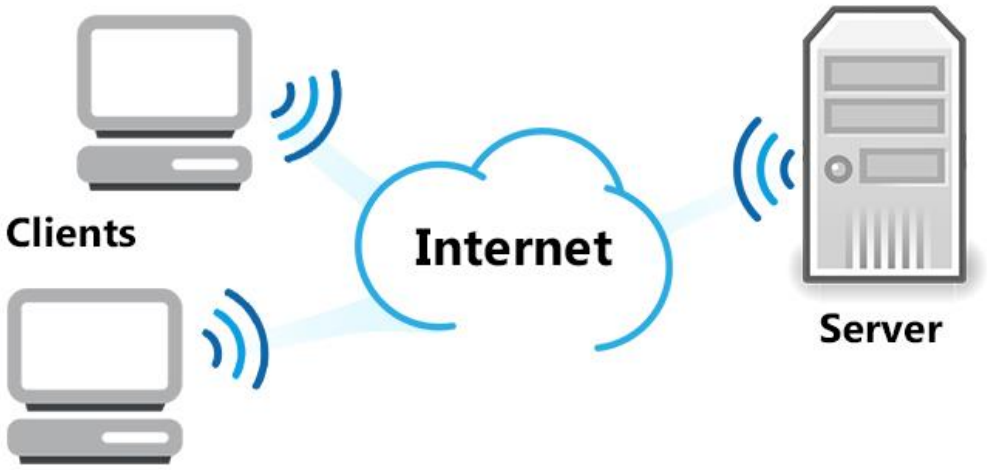

Рисунок 3.1 – Приклад архітектури «клієнт-сервер»

Розглянемо переваги та недоліки даної моделі. Переваги:

- покращений обмін даними: дані зберігаються та обробляються на сервері за допомогою звичайних процесів, доступних для визначених користувачів (клієнтів) через дозволений доступ. Використання структуризованої мови запитів підтримує зручний інтерфейс для клієнтів, а також прозорість процесу обміну даними між користувачами;
- спільні ресурси для різних платформ: програми, які використовуються для моделі клієнт / сервер, побудовані незалежно від апаратної платформи або технічного походження відповідного програмного забезпечення, що забезпечує відкрите обчислювальне середовище, надаючи можливість користувачам отримувати послуги клієнтів і серверів (баз даних, додатків, серверів зв'язку);
- можливість обробки даних, незважаючи на місце розташування: архітектура «клієнт – сервер» орієнтована, насамперед, на користувачів системи. Тому, однією з основних її переваг є надання користувачам можливості безпосередньо увійти в систему незалежно від місця їх розташування;
- просте обслуговування: оскільки архітектура «клієнт – сервер» є моделлю, що представляє собою розподілені обов'язки між незалежними пристроями, інтегрованими в мережу, це є перевагою з точки зору

обслуговування. Замінити, відремонтувати, оновити та перемістити сервер досить просто, поки клієнти залишаються незадіяними;

- безпека: сервери мають кращий контроль доступу та ресурсів, щоб гарантувати, що тільки авторизовані клієнти можуть отримувати доступ або маніпулювати даними, а серверні оновлення ефективно адмініструються.

Недоліки:

- перевантажені сервери: коли є багато одночасних запитів клієнтів, сервери сильно перевантажуються, утворюючи затори трафіку;
- вплив централізованої архітектури: оскільки архітектура «клієнт – сервер» є централізованою, вихід з ладу серверу робить неможливим виконання запитів клієнтів.

З наведеного вище можна зробити підсумок, що мережева модель клієнт / сервер, не зважаючи на свої недоліки, задовольняє поставлені до неї вимоги по орієнтованості на користувача, безпеку та відповідність концепції системи, що розробляється у даному проєкті.

3.2 Мікро фреймворк Flask

В даній роботі для написання власного серверу було використано мікро фреймворк Flask. Він являє собою легковаговий WSGI фреймворк для веб-додатків, написаних на мові Python. WSGI (Web Server Gateway Interface) – це специфікація, яка описує, яким чином веб-сервер повинен обмінюватися даними з веб-додатками та як веб-додатки можуть бути пов'язані в спільній обробці одного запиту.

Flask класифікується як мікро фреймворк, оскільки не потребує конкретних інструментів чи бібліотек. Flask призначений для швидкого та легкого створення власного серверу з можливістю масштабування проекту. Наразі він є одним з найпопулярніших фреймворків для веб-додатків на мові Python.

Flask надає пропозиції, але не накладає жодних залежностей на проєкт, так розробник може сам обирати інструменти та бібліотеки, які потрібно використовувати. В Інтернеті існує багато розширень, які полегшують додавання нових функціональних можливостей Існують, наприклад, розширення для об'єктно-реляційних картографів, перевірки форм, обробки завантажень, різних технологій відкритої аутентифікації та кількох загальних інструментів, пов'язаних з самим фрейморком.

Перевагами даного мікро фреймворку є:

- наявність серверу розробки та відлагоджувача;
- комплексна підтримка модульного тестування (unit testing);
- підтримка безпечних файлів cookie (клієнтської сесії);
- підтримка Unicode;
- сумісність з Google App Engine;
- RESTful відправлення запитів;
- використання шаблонів Jinja2;
- докладна документація;
- наявність розширень для підключення бажаної функціональності;

На рисунку 3.2 показано створення найпростішого веб-додатку з використанням мікро фреймворку Flask. При звертанні до даного веб-серверу буде отримана відповідь у вигляді фрази «Hello world!».

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

Рисунок 3.2 – Веб-додаток, створений за допомогою мікро фреймворку Flask

3.3 Веб-API та мережевий протокол REST

У даному проєкті передача даних між клієнтом та сервером відбувається за допомогою використання мережевого протоколу REST та через веб-API. Для розуміння роботи додатку потрібно розібрати даний інструмент.

Насамперед, перед створення веб-додатку важливо мати чітке розуміння того, що собою являє API. API (програмний інтерфейс додатку) - це набір програмного коду, який дозволяє виконувати обмін даними між одним програмним продуктом та іншим. Крім цього, API також містить умови цього обміну. На рисунку 3.3 показано, як саме API працює.

API складається з двох компонентів:

1. Технічної специфікації, що описує варіанти обміну даними між програмами, при цьому специфікація виконується у вигляді запиту на обробку та протоколів доставки даних.

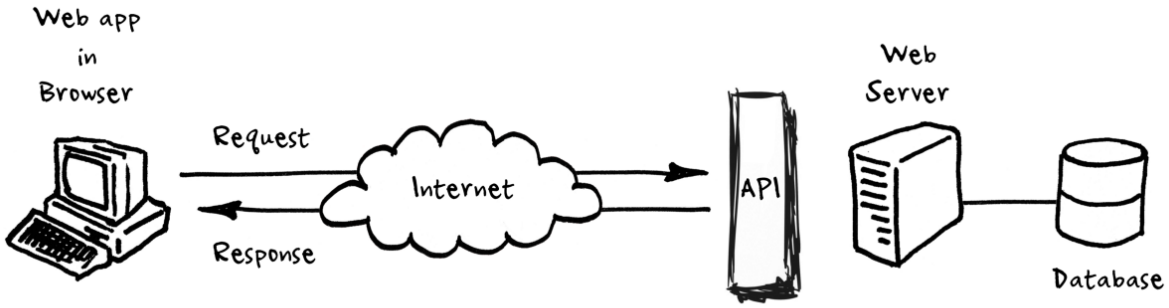

Рисунок 3.3 Візуалізація роботи програмного інтерфейсу API

2. Інтерфейсу програмного забезпечення, записаного у специфікацію, яка його представляє.

Як приклад функціонування API можна навести наступну ситуацію: певне програмне забезпечення потребує доступу до інформації, що його цікавить (наприклад, ціни на готельний номер X на певні дати) або до функціональності (наприклад, прокласти маршрут від точки A до точки B на карті на основі місця розташування користувача) з іншого програмного

забезпечення. В такому випадку, воно викликає метод зі свого API, відповідно до вимог щодо потрібних даних / функціональних можливостей та звертається до API потрібного програмного забезпечення. Те, в свою чергу, повертає ці дані / функціональні можливості, які вимагає інша програма. Як можна побачити, інтерфейс, через який ці два додатки спілкуються, - це саме те, що називається API.

API слугують численним цілям. Як правило, вони можуть спростити та пришвидшити розробку програмного забезпечення. Розробники можуть додати до існуючих рішень функціональні можливості (наприклад, механізм рекомендування, бронювання проживання, розпізнавання зображень, обробки платежів) від інших постачальників або побудувати нові програми, використовуючи послуги сторонніх постачальників. У всіх цих випадках фахівцям не доведеться мати справу з сирцями, намагаючись зрозуміти, як працює те чи інше рішення. Вони просто підключають своє програмне забезпечення до іншого. Іншими словами, API служать рівнем абстракції між двома системами, приховуючи їх складність та робочі деталі.

В даному проєкті використовується клас веб-API, котрий, доречі, є найпоширенішим. Веб-API забезпечує машино-читабельну передачу даних та функціональних можливостей між веб-системами, що представляють архітектуру «клієнт – сервер». Ці API в основному доставляють запити веб-додатків та відповіді від серверів за допомогою протоколу передачі гіпертексту (HTTP). Розробники можуть використовувати веб-API для розширення функціональності своїх додатків або сайтів. Наприклад, API Pinterest поставляється з інструментами для додавання даних користувачів на веб-сайт.

Отже, метою специфікацій API є стандартизація обміну даними між веб-сервісами. У цьому випадку стандартизація означає здатність різноманітних систем, написаних різними мовами програмування та / або запущених на різних ОС чи з різними технологіями, безперебійно спілкуватися між собою. Підхід, який буде використаний при цьому в проєкті, описаний далі.

REST (REpresentational State Transfer) – це архітектурний стиль програмного забезпечення, який визначає набір обмежень, які будуть використані для створення веб-служб. Іншими словами це набір правил, яких розробники дотримуються під час створення свого API. Одне з цих правил передбачає, що користувачі повинні мати можливість отримати фрагменти даних, коли вони посилаються запит на певну URL-адресу. Кожна URL-адреса називається запитом, тоді як дані, що надсилаються у зворотньому напрямку, називаються відповіддю.

Як і будь-який інший архітектурний стиль, REST має свої керівні обмеження, які повинні бути виконані, щоб інтерфейс можна було називати RESTful. Ці принципи перераховані нижче:

- клієнт-серверна модель - відокремлюючи проблеми користувальницького інтерфейсу від проблем зберігання даних, покращується портативність користувальницького інтерфейсу на кількох платформах та здатність до масштабування, шляхом спрощення серверних компонентів;
- відсутність запам’ятовування станів - кожен запит від клієнта до сервера повинен містити всю інформацію, необхідну для відповіді на запит. Іншими словами, повинна бути можливість зробити два або більше HTTP-запитів у будь-якому порядку, після чого будуть отримані однакові відповіді;
- кешування – це обмеження вимагає, щоб дані у відповіді на запит неявно або явно позначали як кешовані або не кешовані. Якщо відповідь є кешованою, то для кешу клієнта надається право повторно використовувати ці дані для наступних, еквівалентних запитів;
- уніфікований інтерфейс - За допомогою застосування даного принципу спрощується загальна архітектура системи та покращується прозорість взаємодій;
- багаторівнева система - такий стиль дозволяє архітектурі складатися з ієрархічних рівнів, обмежуючи поведінку компонентів таким чином, що кожен компонент не може "дивитися" за межі рівня, з яким він взаємодіє;

- код на вимогу (необов’язково) - більшу частину часу сервер буде надсилати назад статичні види ресурсів у вигляді XML або JSON. Однак при необхідності сервер може надсилати клієнту код, котрий може виконуватися.

RESTful веб-API представлений 5 типами запитів, що надсилаються на сервер. Вони дозволяють отримувати, надсилати, змінювати та видаляти данні на сервері. Ці запити наведено нижче:

- `GET`. Цей запит використовується для отримання ресурсу з сервера. Якщо виконується запит `GET`, сервер шукає запитувані дані та надсилає їх у вигляді відповіді. Цей метод запиту є запитом за замовчуванням.
- `POST`. Цей запит використовується для передачі нового ресурсу на сервері. Якщо виконується запит `POST`, сервер отримує дані, що передаються, та виконує на ними визначені в коді дії.
- `PUT` і `PATCH`. Ці два запити використовуються для оновлення ресурсу на сервері. Якщо виконується запит `PUT` або `PATCH`, сервер оновлює потрібні дані і повідомляє, чи оновлення пройшло успішно.
- `DELETE`. Цей запит використовується для видалення ресурсу з сервера. Якщо виконується запит `DELETE`, сервер видаляє запис у базі даних і повідомляє, чи це було успішно.

3.4 JSON файли

В попередніх пунктах було описано особливості організації серверу та його API, однак є ще одна важлива річ – формат даних, в якому вони передаються від клієнта до сервера і навпаки. В даному проєкті використовується формат передачі даних JSON.

JSON (JavaScript Object Notation) - стандартний текстовий формат для представлення структурованих даних на основі синтаксису об'єкта JavaScript. Незважаючи на походження, JSON – це незалежний від мови програмування формат даних, так як багато сучасних мов містять у собі код для генерування та

аналізу даних у даному форматі. Він зазвичай використовується для передачі даних у веб-додатках (наприклад, відправка даних з серверу на клієнт, або навпаки).

У JSON визначено лише дві структури даних: об'єкти та масиви. Об'єкт - це набір пар ім’я-значення, а масив - список значень.

Основними типами даних, що використовуються у даному форматі (крім вище згаданих) є:

- число: знакове десяткове число, яке може містити дробову частину і може використовувати експоненціальне позначення E, але не може включати в себе не-числа, такі як NaN;
- рядок: послідовність з нуля або більше символів у кодуванні Unicode. Рядки розмежовуються подвійними лапками і підтримують синтаксис escape-послідовностей;
- булеве значення: true або false;
- null: порожнє значення, для позначення використовується слово null.

Приклад формату даних JSON зображений на рисунку 3.4:

```
{
   "firstName": "Duke",
   "lastName": "Java",
   "age": 18,
   "streetAddress": "100 Internet Dr",
   "city": "JavaTown",
   "state": "JA",
   "postalCode": "12345",
   "phoneNumbers": [
      { "Mobile": "111-111-1111" },
      { "Home": "222-222-2222" }
   ]
}
```

Рисунок 3.4 – приклад даних у форматі JSON

Судячи з нього, можна виділити наступні правила синтаксису даного формату:

- об'єкти оголошуються фігурними дужками ({}), їхні пари ім’я-значення розділені комою (,), а безпосередньо ім'я та значення в парі розділені двокрапкою (:). Імена в об'єкті - це строки, тоді як значення можуть бути будь-якого з семи типів даних, включаючи інший об'єкт або масив;
- масиви оголошуються квадратними дужками ([]), а їх значення розділені комою (,). Значення в масиві можуть бути різного типу, включаючи інший масив або об'єкт;
- коли об'єкти та масиви містять в собі інші об'єкти чи масиви, дані мають деревоподібну структуру.

Як підсумок, даний формат ідеально підходить для обміну даними між клієнтом та сервером завдяки тому, що він легко читається, пишеться і є більш компактний, ніж інші, подібні формати. Саме тому він набув такої широкої популярності та, крім того, був обраний для виконання його прямої задачі у даному проєкті.

3.5 База даних PostgreSQL

Деяку інформацію зручніше зберігати у спеціально розроблених для цього місцях, таких як бази даних. Дана опція використовується в проєкті, так як організація даних у такий спосіб є зручнішою, надійнішою та більш легкою для сприйняття.

Поняття бази даних означає організований набір структурованої інформації або даних, що зазвичай зберігаються в електронному вигляді в комп'ютерній системі. База даних, як правило, контролюється системою управління базами даних (СУБД).

Інформація в найпоширеніших типах баз даних, що діють сьогодні, зазвичай моделюються в рядках та стовпцях у серії таблиць, з ціллю зробити

обробку та запити даних більш ефективними. З даною організацією можна легко отримувати доступ, керувати, змінювати, оновлювати, контролювати та організовувати дані. Більшість баз даних використовують структуровану мову запитів (SQL) для запису та запиту даних.

PostgreSQL - це вільна та відкрита система управління реляційними базами даних. Вона вимагає мінімальних зусиль для її підтримки, що є перевагою відносно її аналогів. Для більш комфортного керування базою даних можна використати програмне забезпечення pgAdmin. Воно надає користувацький інтерфейс для роботи з базою даних PostgreSQL, що полегшує роботу з нею та дозволяє швидше проводити аналіз її стану та проводити з нею будь-які потрібні маніпуляції в наочному вигляді.

Як підсумок, PostgreSQL надає всі потрібні інструменти для зручного зберігання даних і задовольняє вимоги, поставлені до бази даних в даному проєкті, тому вибір впав саме на неї.

4. ОПИС ПРОГРАМНОЇ СТРУКТУРИ КІНЦЕВОГО ПРОДУКТУ

4.1 Опис програмної структури клієнтського додатку

Як вже було сказано на початку (див. 1.4), клієнтська частина системи формування та контролю електронної черги для мобільних пристроїв являє собою додаток на базі операційної системи Android.

В ОС Android основними компонентами будь-якого додатку є layouts та Activities. Перші – це XML файли, що містять у собі розмітку інтерфейсу відповідної сторінки з розміщенням певних елементів на ній (таких як кнопки, повзунки, місця для введення/виведення тексту, тощо), а другі реалізують логіку взаємодії користувача з елементами, описаними у layouts файлах.

Додаток, що розроблявся у даному проєкті, має наступні layouts файли:

- activity_main.xml – для розмітки вікна головного меню додатку;
- activity_root_interface.xml – для розмітки інтерфейсу головного меню додатку;
- activity_user_interface.xml – для розмітки користувацького інтерфейсу;
- activity_my_queues.xml – для розмітки інтерфейсу вікна черг користувача;
- activity_settings.xml – для розмітки інтерфейсу вікна налаштувань додатку;
- dialog_connect_queue.xml – для розмітки інтерфейсу діалогового вікна підключення до черги;
- dialog_authorization.xml – для розмітки інтерфейсу діалогового вікна авторизації у додатку;
- dialog_create_queue.xml – для розмітки інтерфейсу діалогового вікна підключення до черги;
- queue_users_list_element.xml – для розмітки інтерфейсу елементу списку користувачів у черзі;

– user_queues_list_element.xml – для розмітки інтерфейсу елементу списку черг користувача;

Так як layouts файли є досить простими, буде розглянуто лише файл queue_users_list_element.xml (рисунок 4.1):

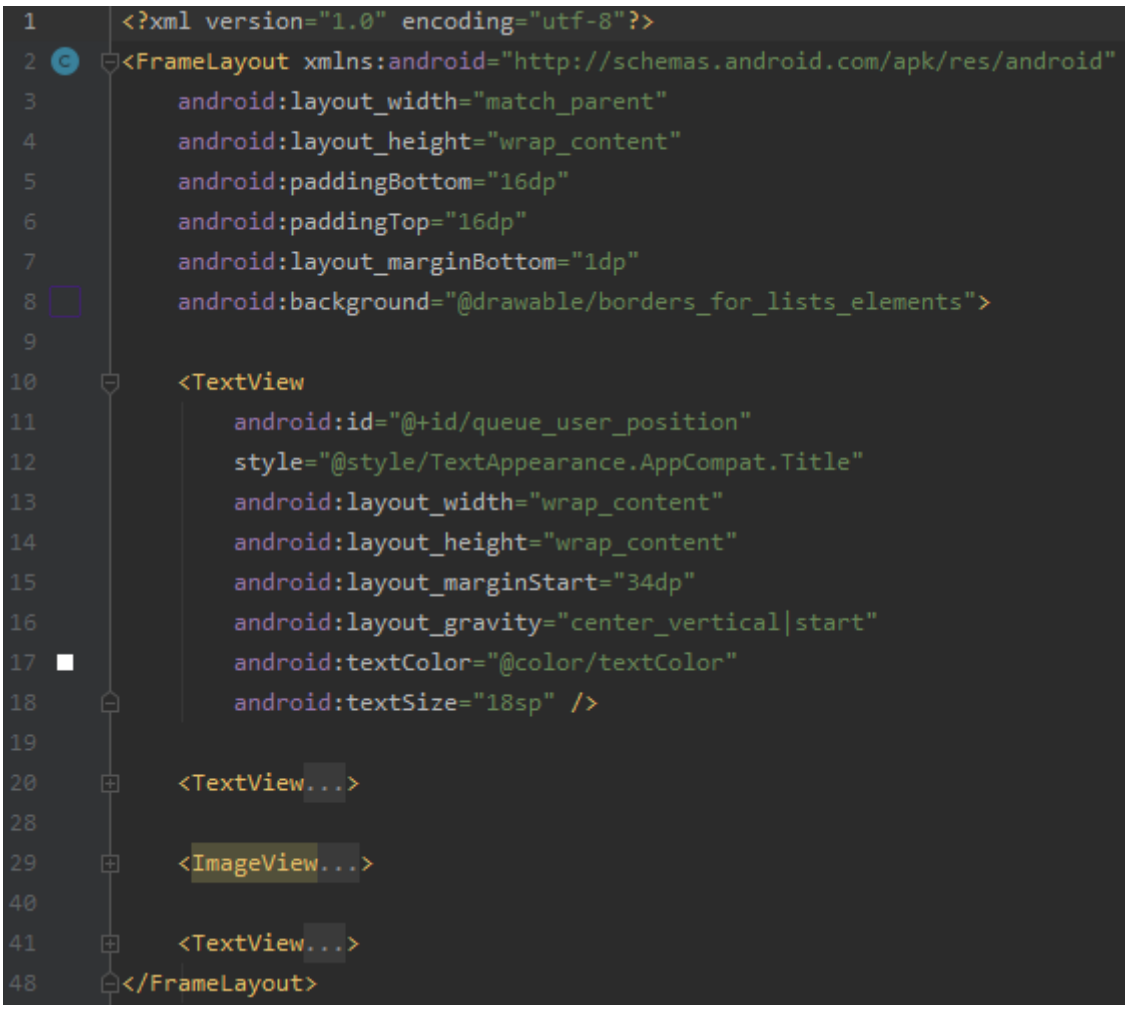

Рисунок 4.1 – XML файл для розмітки інтерфейсу елементу списку користувачів у черзі

В першому рядку файлу вказується його версія та тип кодування символів. Далі оголошується тип макету (layout), який буде кореневим. В даному випадку це FrameLayout. Існують різні типи макетів, кожен з яких має свої особливості. Даний макет відрізняється тим, що об’єкти можуть накладатися один на одного.

Макет слугує об’єктом, в якому потім будуть розміщені елементи, що оголошені всередині нього (такі як TextView, ImageView та т.д.). Кожен з них

представляє певний елемент у вікні користувача. Наприклад, TextView відповідає за відображення тексту на екрані, в той час як ImageView – за відображення картинок.

Кожен XML об’єкт має певний набір атрибутів, що повністю описують його. Їх існує велика кількість, наприклад, це може бути ширина, висота, видимість, колір, форма об’єкту, тощо. XML об’єкти можуть мати як спільні, так і відмінні атрибути. Їх можна вказувати як вручну так і програмно.

Для перевірки розмітки описаного XML файлу в Android Studio існує спеціальна функція «Blueprint» (рисунок 4.2), що дозволяє побачити визначений макет у вигляді креслення.

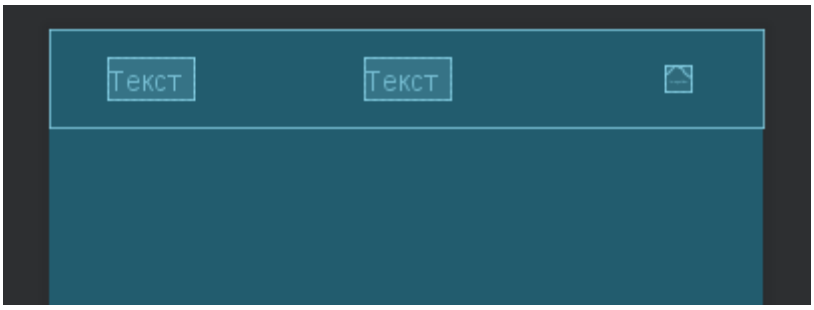

Рисунок 4.2 – Функція «Blueprint» в Android Studio для показу XML розмітки файлу

Інші XML файли даного проєкту мають аналогічну структуру.

Щодо програмної частини клієнтського додатку, написаної мовою Java, то вона складається з набору класів, частина з яких є нащадками класу Activities, а інша – допоміжними класами, для розділення логіки додатку.

Таким чином, даний додаток, створений на базі операційної системи Android, складається з наступних класів (класи, що в своїй назві містять слово «Activity» відносяться до Activities):

- MainActivity – реалізовую логіку головного меню додатку;
- SettingsActivity – реалізовує логіку меню налаштувань додатку;
- MyQueuesActivity – реалізовує логіку вікна представлення черг користувача;

– RootInterfaceActivity – реалізовує логіку сторінки адміністратора черги;

– UserInterfaceActivity – реалізовує логіку сторінки користувача черги;

– ServerConnector – реалізовує логіку взаємодії додатку з сервером;

– MyQueuesListAdapter – реалізовує логіку адаптеру списку черг користувача;

– QueueUsersListAdapter – реалізовує логіку адаптеру списку користувачів черги;

Клас MainActivity складається з методів, що реагують на натиски відповідних пунктів меню на екрані (рисунок 4.3), а також з методів отримання даних від користувача (таких як його ім'я та MAC-адреса, що використовуються надалі для його ідентифікації).

```
23  public class MainActivity extends AppCompatActivity {
24      public static final String PREFS_NAME = "PrefsFile";
25
26      @Override
27      protected void onCreate(Bundle savedInstanceState) {...}
37
38      @SuppressLint("ResourceAsColor")
39      public void onShowNameDialog(Bundle savedInstanceState) {...}
97
98      public boolean checkIfUserNameEmpty () {...}
106
107     public void setMacAddr (){...}
114
115     public static String getMacAddr() {...}
141
142     public void onCreateQueue(View view){...}
150
151     public void onConnectQueue(View view){...}
155
156     public void onMyQueues(View view){...}
160
161     public void onSettings(View view){...}
165 }
```

Рисунок 4.3 – Структура класу MainActivity

Клас SettingsActivity складається з методів, що реагують на зміни налаштувань додатку, після шляхом натиском на відповідні елементи на екрані користувачем (такі як кнопки або перемикачі). Прикладом цього є частина коду, що вмикає / вимикає аудіо потік додатку (рисунок 4.4):

```
if (volumeMode) {
    AudioManager amanager = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
    amanager.setStreamMute(AudioManager.STREAM_NOTIFICATION, state: false);
    Toast toast = Toast.makeText(getApplicationContext(),
            text: "Turned on volume.", Toast.LENGTH_SHORT);
    toast.show();
} else {
    AudioManager amanager = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
    amanager.setStreamMute(AudioManager.STREAM_NOTIFICATION, state: true);
    Toast toast = Toast.makeText(getApplicationContext(),
            text: "Turned off volume.", Toast.LENGTH_SHORT);
    toast.show();
}
```

Рисунок 4.4 – Частина коду, що вмикає /вимикає аудіо потік додатку

В даній частині коду створюється об’єкт класу AudioManager та за допомогою його методу setStreamMute вимикається аудіо потік. Після цього, за допомогою об’єкту класу Toast на екран виводиться відповідне повідомлення.

Клас ServerConnector містить у собі методи, котрі встановлюють зв’язок з сервером та надсилають на нього відповідні запити, повертаючи його відповідь. Крім методів даний клас містить статичні поля типу String, котрим присвоєно URL-адреси для відповідних запитів. Наприклад, поле CREATE_QUEUE даного класу має значення «api/methods/queue/create», що є частиною URL-адреси, котра відповідає за створення черги на сервері.

Класи QueueUsersListAdapter та MyQueuesListAdapter виконують роботу одного й того ж характеру лише з різними об’єктами, тому вони є дуже схожими. Головним їх призначенням є контроль відображення даних, отриманих з серверу, на відповідних вікнах додатку.

Класи MyQueuesActivity своєю головною задачею має вивід активних черг користувача. Для цього використовується об’єкт типу RecyclerView та адаптер MyQueuesListAdapter, що був описаний вище.

Класи RootInterfaceActivity та UserInterfaceActivity відповідають за логіку роботи вікон адміністратора та користувача відповідно. Однією з особливостей даних класів є те, що вони використовують об'єкти типу Runnable для періодичного надсилання запитів на сервер з метою отримання актуальної інформації (рисунок 4.5).

```
private Handler handler = new Handler();
private Runnable periodicUpdate = () -> {
        handler.postDelayed(periodicUpdate, delayMillis: 10*1000);
        getQueueQuery();
};
```

Рисунок 4.5 – Об'єкт класу Runnable для періодичного надсилання запитів на сервер

Об'єкт типу Handler створюється для того, щоб вказувати точний час затримки між запитами за допомогою функції postDelayed().

Для формування запитів на сервер використовується клас AsyncTask, що реалізовує окремі потоки в окремій програмі. Це дозволяє не блокувати роботу з додатком під час очікування відповіді від серверу та збільшити комфорт використання додатку для користувача.

Крім файлі типу layouts та Activities клієнтський додаток має спеціальні файли ресурсів, що знаходяться у папці res проєкту додатку. Вони являють собою XML файли з описи нами в них сталими значеннями, що використовуються у кількох місцях програми для відображення у інтерфейсі. Це можуть бути рядки, числа, стилі, картинки, тощо.

Все вище наведене в сукупності являється ядром додатку. Воно реалізовує логіку роботи додатку та інтерфейс, який бачить кінцевий користувач даного продукту. Клієнтський додаток на основі ОС Android слугує проміжною ланкою між користувачем та сервером, надаючи зручний інтерфейс взаємодії з останнім. розроблюваної системи формування та керування електронної черги для мобільних пристроїв описана в наступному підрозділі.

4.2 Опис програмної структури серверної частини

Серверна частина розроблюваної системи формування та керування електронної черги для мобільних пристроїв складається з наступних компонентів:

- модуль REST-ful API;
- модуль серверної логіки додатку;
- модуль для взаємодії з базою даних;
- модуль бази даних;
- місце для локального зберігання даних;

Написання серверної частини виконувалося мовою Python, так як обраний мікро фреймворк Flask розроблений саме для неї.

Модуль REST-ful API. Даний модуль зберігається у файлі «server/api/api.py» даного проєкту. Він відповідає за запуск серверу та містить у собі його програмний мережевий інтерфейс. За допомогою перших стрічок даного файлу (рисунок 4.6) до нього підключаються всі необхідні модулі. Так, наприклад, перша стрічка коду дозволяє підключити до даного файлу всі компоненти, що реалізовані в мікро фрейморці Flask та потрібні для його роботи, а друга стрічка коду підключає модуль «json» для подальшої роботи з даним форматом.

```
1  import flask
2  import json
3  from app.app import QueueManipulation
4  from flask import request, jsonify
5
6  app = flask.Flask(__name__)
7  app.config["DEBUG"] = True
8
9  app.run(host="0.0.0.0")
```

Рисунок 4.6 – Код створення серверу

Після цього, йде створення безпосередньо серверу, строкою «app = flask.Flask(__name__)». Даний код створює об’єкт класу «Flask», що реалізовує

веб-додаток (тобто, сервер) та передає йому змінну, що зберігає ім’я даного файлу. Це потрібно для того, щоб у разі, коли даний файл є виконуваним, програма запускала відповідні процеси. Для старту роботи серверу достатньо запустити функцію app.run(). У разі передачі до неї параметру «DEBUG=True» сервер запуститься у режимі налагодження і всі взаємодії з ним та виводи програми будуть паралельно виводитися у консоль. Даний режим зручним для виявлення несправностей у роботі серверу та їх усунення.

Далі в даному файлі описуються безпосередньо методи обміну даними з сервером та дії, які він буде виконувати при надходженні відповідних URL запитів. Прикладом коду одного з методів (а саме, створення черги) веб-API даного додатку є рисунок 4.7:

```
@app.route('/api/methods/queue/create', methods=['GET'])
def create_queue():
    queue_name = request.args.get('name', False)
    queue_password = request.args.get('password', False)
    user_name = request.args.get('user', False)
    user_mac = request.args.get('mac', False)

    if queue_name and queue_password and user_name and user_mac:
        manipulator = QueueManipulation()
        queue_id = manipulator.create_queue(queue_name, queue_password, user_name, user_mac)
        queue = manipulator.get_queue(queue_id)
        response = {'queue_id': queue_id, 'queue_name': queue_name, 'queue': queue, 'result': 'true'}
    else:
        response = {'result': 'false', 'error': 'Incorrect arguments!'}

    return jsonify(response)
```

Рисунок 4.7 – Код створення серверу

@app.route – це функція декоратор, що визначає оголошену після неї функцію як ту, котра буде запускатися при надходженні на сервер запиту GET за адресою «адреса_серверу / адреса_запиту». На рисунку 4.7 можна побачити, що адреса даного запиту являється рівною «/api/methods/queue/create».

Виклик функції request.args.get(«назва аргументу», «результат повернення у разі невдачі») дозволяє отримати аргументи, що було передано разом з запитом до серверу. В даному випадку це назва черги, що створюється, її пароль, ім’я користувача, що її створює та його унікальний ідентифікатор.

Після цього відбувається перевірка, чи всі елементи було вказано у забиті. Якщо ні, то формується відповідь зі рядком ‘false’ у ключі результату, та

з текстом помилки у графі помилки. Якщо ж всі аргументи, що очікує функція було передано, то створюється об’єкт класу QueueManipulation, що реалізовує логіку веб-додатку, та за допомогою нього викликаються методи для створення черги, отримання її стану та компонується відповідь із зазначеними параметрами у змінній «response». Після того як відповідь було сформовано, вона переводиться у формат JSON за допомогою функції jsonify() та повертається користувачу.

За такою ж логікою було описано й інші методи веб-API даного серверу, що виконують відповідні їм задачі.

Модуль серверної логіки додатку. Даний модуль зберігається у файлі «server/app/app.py» даного проєкту. Його призначенням є оголошення класу, що реалізує повний набір методів для маніпуляції з чергами та відображенням даних дій на базі даних. Цей клас має назву QueueManipulation. В ньому реалізовані такі функції як:

- створення черги на сервері;
- генерація унікального ідентифікатору для створеної черги;
- видалення черги;
- додавання користувача до черги;
- видалення користувача з черги;
- видачі стану черги;
- вивантаження / зчитування черг на / з фізичний (-го) пристрій (-ою);
- зв’язок з базою даних;

На рисунку 4.8 зображено конструктор даного класу, в якому створюються основні його поля, такі як:

- self.dp – поле класу, що містить шлях до файлу, в якому зберігаються безпосередньо черги з користувачів;
- self.queue_dict – поле класу, що зберігає у собі черги користувачів після зчитування даної інформації з файлу;
- self.db – поле класу, що містить у собі об’єкт класу QueueDatabase для взаємодії з базою даних серверу.

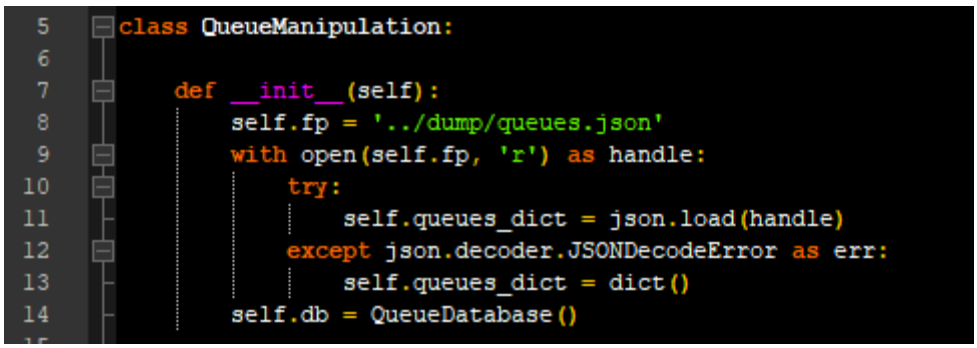

Рисунок 4.8 – Конструктор класу QueueManipulation

Даний конструктор при створенні об’єкту класу QueueManipulation зчитує дані про черги у змінну «self.queue_dict», переводячи при цьому її у об’єкт типу словник мови Python та ініціалізує змінну «self.db» для забезпечення зв’язку з базою даних. Після цього клас є готовим до проведення маніпуляцій з чергами додатку. Будь-яка зміна стану черг одразу відображається на базі даних серверу та файлі локального сховища.

Модуль взаємодії з базою даних. Даний модуль знаходиться у файлі «server/database/database.py». Основною його ціллю є взаємодія з сервером бази даних PostgreSQL. Для цього в ньому реалізовано клас QueueDatabase, що містить у собі описи методів, котрі дозволяють за допомогою бібліотеки «sqlalchemy» надсилати на виконання стандартні запити до бази даних, такі як SELECT, INSERT, UPDATE та DELETE. Кожному з них в даному класі існує програмний відповідник, що дозволяє правильно сформувати запит та передати у нього потрібні параметри, після чого виконати його.

При створенні об’єкту даного класу одразу відбувається підключення до бази даних за наступними строками коду (рисунок 4.9):

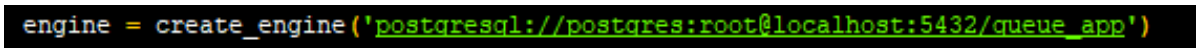

Рисунок 4.9 – Підключення до бази даних за допомогою класу QueueDatabase

Змінна «engine» містить в собі так званий «двигун» для роботи з базою даних, отриманий шляхом виконання бібліотечної функції create_engine() та передачі їй назви бази даних (тут, «postgresql»), імені користувача (тут,

«postgres») та паролю (тут, «root») після чого йде вказання адресу підключення, що є стандартним для даної бази даних (тут, «localhost:5432»). Ще одним важливим параметром в даному рядку є назва бази даних, до якої йде підключення, а саме,в даному випадку, це «queue_app».

Модуль бази даних. Модулем бази даних являється база даних на сервері PostgreSQL, до якої відбуваються звертання. На ньому знаходить база даних «queue_app», в якій реалізована таблиця «queues» для збереження такої інформації про черги як:

- назва черги;
- ідентифікатор черги;
- пароль черги;
- ім’я адміністратора черги;
- ідентифікатор адміністратора черги.

Даний модуль відповідає за збереження даних, що з’являються у наслідок роботи системи формування та керування електронної черги для мобільних пристроїв.

Місце для локального зберігання даних. Від зберігання самих черг у базі даних було прийнято рішення відмовитися через занадто часті звернення за даною інформацією та більш зручну маніпуляції з ними у рамках програми. Тому, було вирішено зберігати дані про стани черг на локальній машині серверу, у файлі «server/dump/queues.json». Дані зберігаються у форматі JSON і зчитуються та оновлюються при будь-якій маніпуляції з чергами у клієнтському додатку.

Всі вище описані модулі складають єдину структуру серверу розроблюваної в даному проєкті системи формування та керування електронної черги для мобільних пристроїв.

5. ТЕСТУВАННЯ РОБОТИ КІНЦЕВОГО ПРОДУКТУ

Після того, як кінцевий продукт було створено, потрібно провести тестування його роботи. Для цього можна використати як спеціальні програми, що запам’ятовують натиски на певні місця екрану та порівнюють отриманий результат з бажаним, або ж вручну. Головною метою тестування є перевірити, наскільки додаток виконує свої основні функції, тому тестування буде проводитися вручну, моделюючи взаємодію користувача з додатком.

Варто зазначити, що тестування проводиться на мобільному пристрої Xiaomi Redmi Note 5, що має версію ОС Android 9. Під час встановлення додатку жодних проблем не виникло і він готовий до роботи (рисунок 5.1).

Рисунок 5.1 – Іконка встановленого додатку

Після встановлення додатку на мобільний пристрій він є готовим до подальшої роботи. По відкриттю додатку з’являється діалогове вікно авторизації що пропонує користувачу ввести його ім’я (рисунок 5.2).

Даний етап неможливо пропустити, так як ім’я користувача є необхідною інформацією про нього. Воно потрібне для візуальної ідентифікації та відображення користувача у списках учасників черг. Жодні маніпуляції, такі як натиски повз форму чи спроба вийти назад, не допомагають уникнути діалогу введення даних.

Загалом існує 2 випадки введення некоректної інформації:

- ім’я занадто довге (довше 35 символів);
- введено пустий рядок.

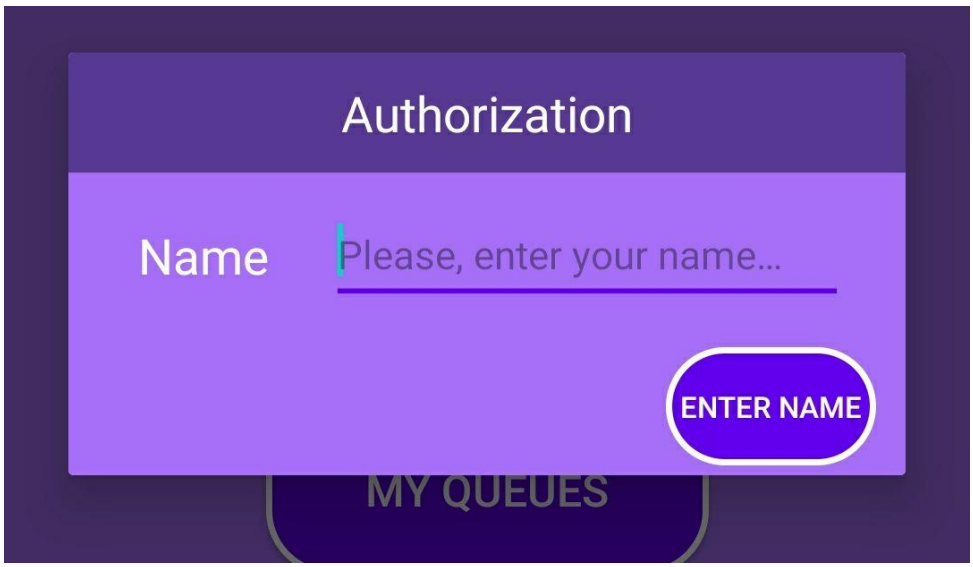

Рисунок 5.2 – Діалогове вікно авторизації при запуску додатку

Введення некоректної інформації спричиняє появу відповідного повідомлення. Наприклад, при введенні пустого рядка користувач побачить наступне повідомлення знизу екрану (рисунок 5.3):

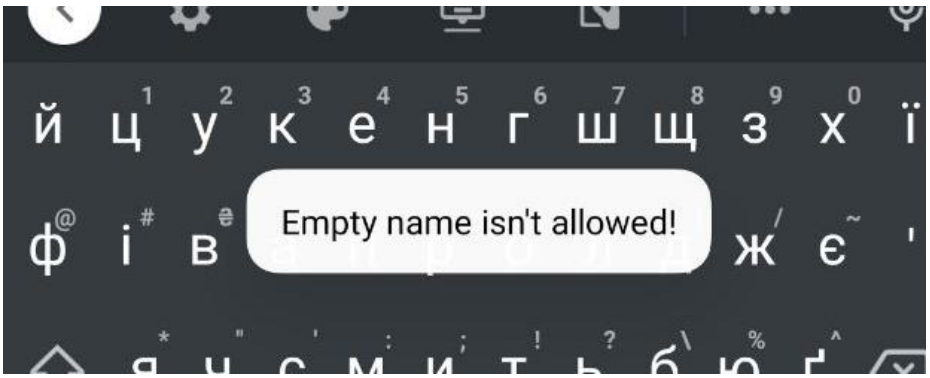

Рисунок 5.3 – Повідомлення про некоректно введене ім’я

Після цього користувач має змогу змінити ім’я та знову спробувати ввести його. Діалогове вікно авторизації зникає по успішному її проходженні.

Після того як дані було введено коректно (при тестуванні було використано ім’я «Sasha») для користувача відкривається головне меню (рисунок 5.4). Воно складається з чотирьох елементів та пропонує наступні опції:

- Create queue – надає можливість створити електронну чергу;

- Connect queue – надає можливість приєднатися до електронної черги;
- My queues – надає можливість переглянути, учасником / керівником яких черг користувач являється;
- Settings – надає певні опції налаштування додатку.

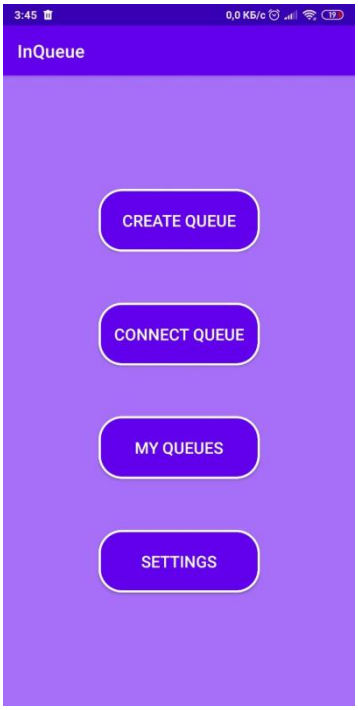

Рисунок 5.4 – Головне меню додатку

При натиску на кнопку «Settings» відбудеться перехід до вікна налаштувань додатку (рисунок 5.5). Воно дозволяє здійснити наступні дії:

- змінити ім’я користувача;
- увімкнути/вимкнути звук;
- увімкнути/вимкнути сповіщення;
- повернутися до головного меню;

Кожній з заданих дій відповідає відповідне поле англійською мовою. При натиску на кнопку «Change name» поле поряд з написом «Your name» стає вільним для редагування. Для зміни імені діють такі ж правила створення імені, що і при авторизації. Після вводу нового імені потрібно натиснути на кнопку «Change name» знову для завершення зміни імені.

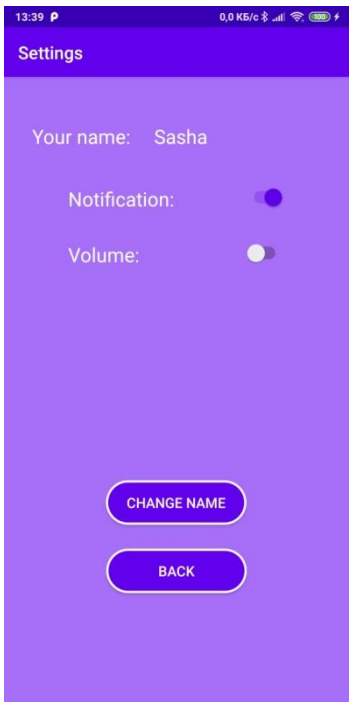

Рисунок 5.5 – Вікно налаштувань додатку «Settings»

В даному випадку було введено ім’я «Oleksandr Kranchuk» та натиснуто на кнопку «Change name». Результатом, отриманий по завершенню зміни імені можна побачити на рисунку 5.6:

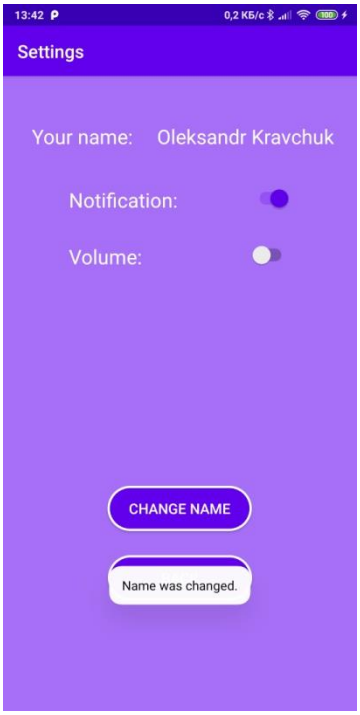

Рисунок 5.6 – Результат зміни імені у вікні налаштувань «Settings»

Важливим моментом при цьому є наявністю підключення до мережі Інтернет та активного серверу, інакше у відповідь буде отримано наступне повідомлення про помилку (рисунок 5.7):

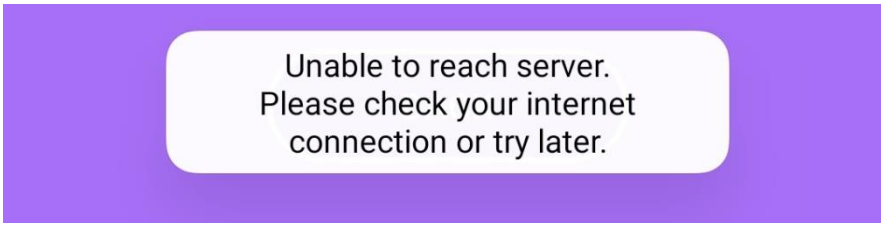

Рисунок 5.7 – Повідомлення про помилку при зміні імені користувача

Дане обмеження не стосується увімкнення/вимкнення звуку/сповіщень, так як це є виключно персональні налаштування користувача.

При натиску на кнопку «Create queue» користувач отримує можливість створити власну електронну чергу. Для цього потрібно придумати ім’я нової черги та пароль і ввести ці дані у відповідні поля діалогового вікна створення черги, що з’явилося (рисунок 5.8):

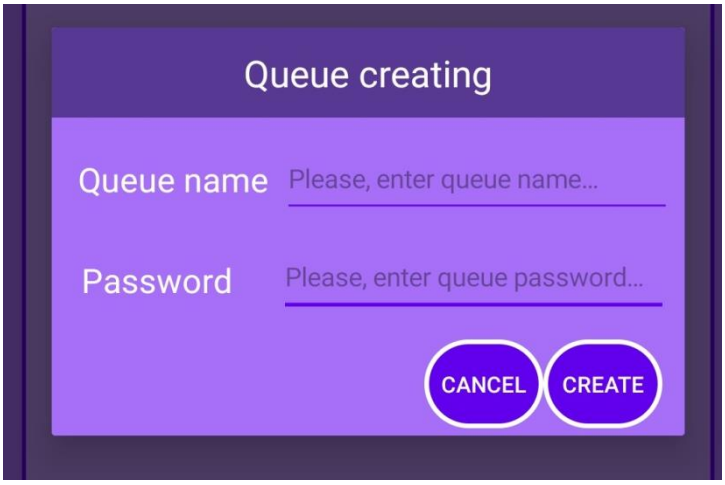

Рисунок 5.8 – Діалогове вікно створення електронної черги

Довжина імені черги повинна бути не менше одного та не більше 50 символів, в той час як довжина паролю повинна бути не менше одного та не більше 20 символів. При недотриманні даних умов черга створена не буде і користувач отримає відповідне повідомлення про помилку на екран. Обмеження на мову вводу даних не має. Даний етап неможливо пропустити.

При натиску на клавішу «Cancel» у діалоговому вікні відбувається повернення до головного меню. При натиску на клавішу «Create», за умови коректно введених даних у поля імені черги та її паролю, діалогове вікно зникає та з’являється вікно адміністратора черги (рисунок 5.9):

Рисунок 5.9 – Вікно адміністратора черги

Дане вікно містить наступні елементи:

- поля «Queue name», «Queue ID» та «People in queue», що справа від себе містять інформацію про чергу – її ім’я, унікальний ідентифікатор та кількість людей у черзі відповідно;
- прямокутну рамку по середині екрану, що є місцем виведення інформації про користувачів, що приєднані до черги (так як зараз немає жодного учасника черги, дана рамка містить надпис «Empty list…»);
- кнопку «Delete queue», що надає можливість видалити дану чергу та повернутися до головного меню;

– кнопку «Refresh», що дає можливість отримати останню інформацію про стан черги з серверу;

– кнопку «Change name», що надає можливість змінити ім’я черги;

– кнопку «Next user», що дозволяє видалити першого користувача з черги та «перейти» до наступного;

– кнопку «Back», що дозволяє закрити дане вікно та повернутися до головного меню;

Слід зазначити, що серед кнопок даного вікна лише кнопка «Back» виконує призначені їй дії за відсутності Інтернет зв’язку / активного серверу. Всі інші кнопки по натиску на них видадуть відповідне повідомлення про помилку, що було показано на рисунку 5.7.

Після того, як до черги було фіктивно приєднано тестовий набір користувачів, дане вікно прийняло наступний вигляд (рисунок 5.10):

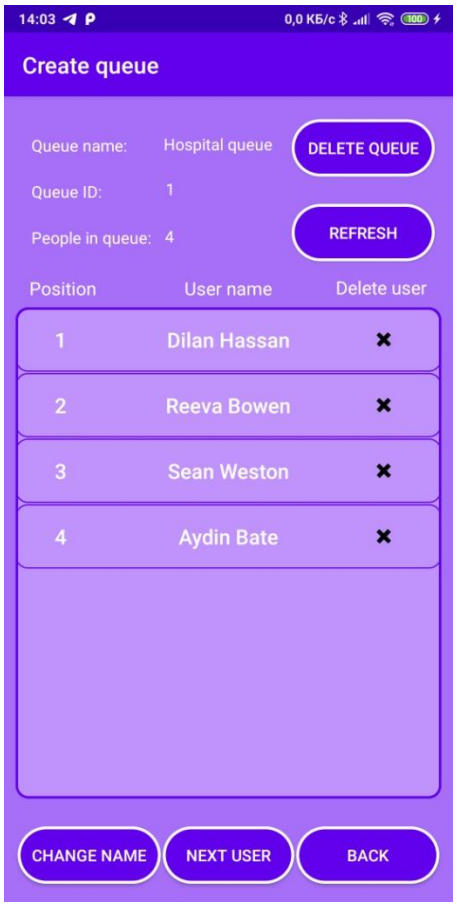

Рисунок 5.10 – Вікно адміністратора черги після приєднання тестового набору користувачів

Можна побачити, що число справа від поля «People in queue» змінилося з 0 на 4, що відповідає актуальній кількості людей в черзі.

Сам список учасників черги складається з таких колонок:

- колонка «Position», що відповідає за позицію користувача у черзі;
- колонка «User name», що відповідає за ім’я користувача;
- колонка «Delete user», що містить кнопку у вигляді хрестика, при натиску на який відповідний користувач видаляється з черги;

Запис будь-якого учасника черги у списку містить кожен з вище перелічених елементів.

При тестуванні видалення користувачів з черги було випробувано 2 способи: за допомогою кнопки «Next user» та за допомогою колонки «Delete user» у списку (для видалення останнього користувача). Результати даних дій наведено на рисунках 5.11 та 5.12 відповідно:

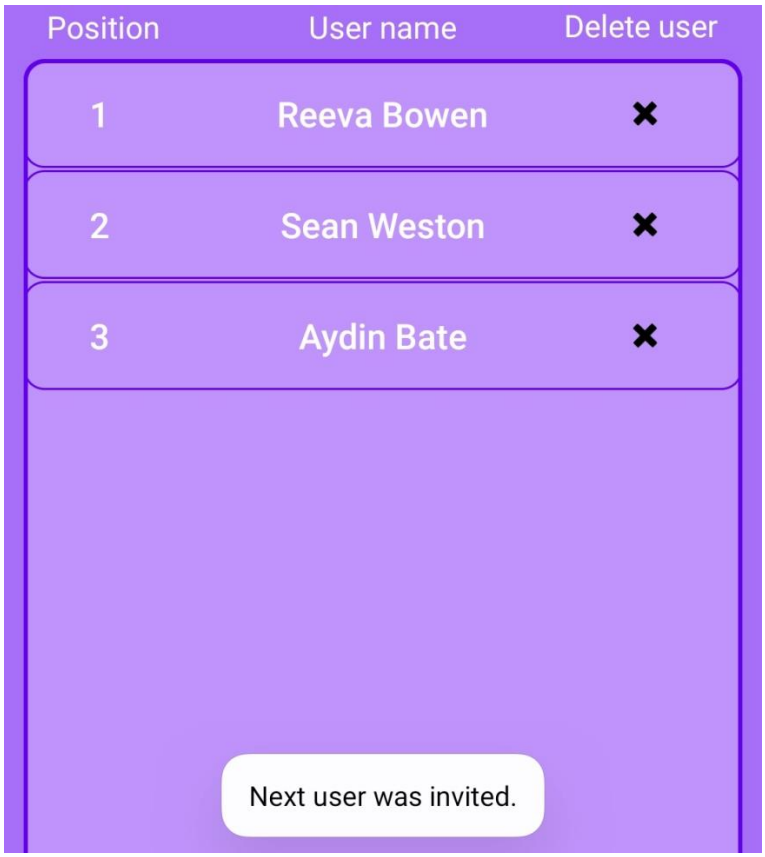

Рисунок 5.11 – Стан списку черги після натиску на кнопку «Next user»

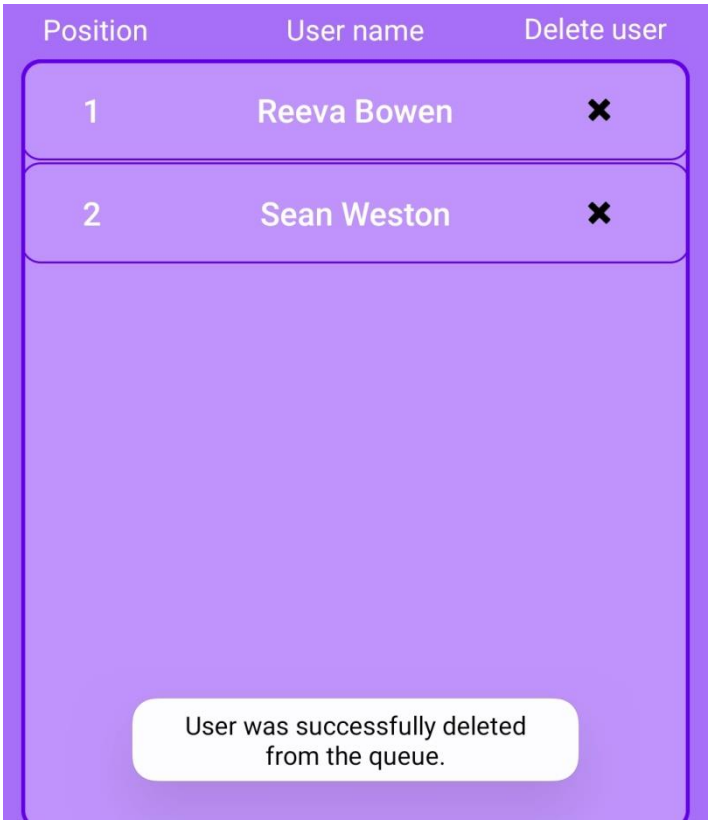

Рисунок 5.12 – Стан списку черги після натиску на хрестик у колонці «Delete user» останнього користувача

Видалення пройшло коректно та дало бажаний результат. При видаленні будь-якого учасника з черги номера учасників, що були у черзі до нього (тобто, з більшим числом у колонці «Position»), зменшуються на одиницю, а позиції інших не змінюються.

Інші кнопки також функціонують відповідно до свого призначення, тому можна повернутися до головного меню за допомогою кнопки «Back».

З головного меню, по натиску на кнопку «Connect queue» з’являється діалогове вікно приєднання до черги (рисунок 5.13). Для підключення до черги потрібно ввести її пароль та ідентифікатор. Повідомлення про помилку спричиняють такі ситуації, як:

- некоректне введення паролю (менше 0 або більше 20 символів);
- некоректне введення ідентифікатору (менше 0 або більше 10 символів);
- введення неіснуючого ідентифікатору черги;

– введення неправильного паролю до існуючої черги;

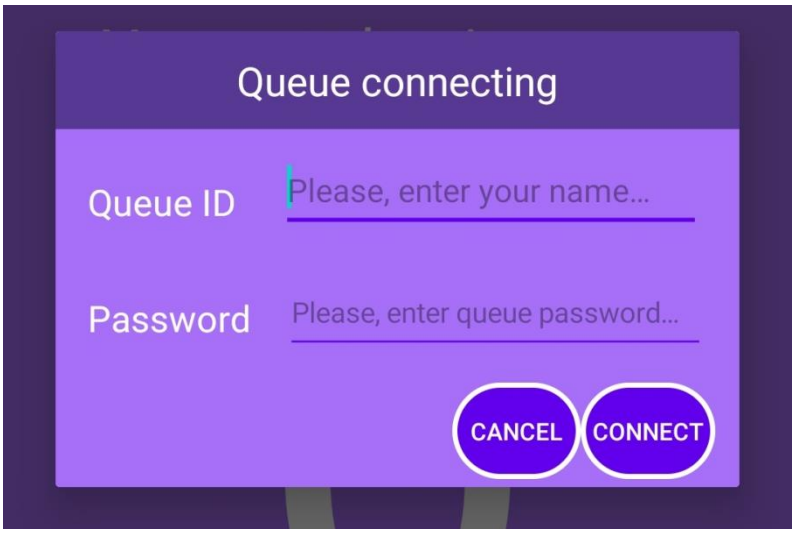

Рисунок 5.13 – Діалогове вікно підключення до черги

При підключенні до існуючої черги користувач потрапляє або у вікно адміністратора черги (якщо він є її власником), або у вікно учасника черги у протилежному випадку.

Вікно учасника черги має наступний вигляд (рисунок 5.14):

Рисунок 5.14 – Вікно учасника черги

Воно містить наступні елементи:

- поля «Queue name», «Queue ID» та «People in queue», що справа від себе містять інформацію про чергу – її ім’я, унікальний ідентифікатор та кількість людей у черзі відповідно;
- число по середині екрану, що є місцем виведення користувача в черзі;
- кнопку «Leave queue», що надає можливість покинути дану чергу та повернутися до головного меню;
- кнопку «Refresh», що надає можливість отримати останню інформацію про стан черги з серверу;
- кнопку «Back», що дозволяє закрити дане вікно та повернутися до головного меню;

Дане вікно оновлюється відповідно до актуальної інформації на сервері та видає сповіщення користувачу, коли його позиція у черзі стає рівною одиниці. Кнопки «Leave queue» та «Refresh» вимагають підключення до Інтернету для коректного їх функціонування.

Останнім вікном, що ще не тестувалося, є вікно, перехід до якого відбувається по натиску на кнопку «My queues» в головному меню. Після виходу до головного меню та натиску відповідної кнопки з’являється вікно черг користувача (рисунок 5.15).

Дане вікно складається з таких елементів:

- списку, що містить колонки «Queue name», «Number of users» та «Administrated», що відповідають за назву черги, кількість людей у черзі та помітку адміністратора (вона з’являється у разі, якщо даний користувач є адміністратором черги) відповідно;
- кнопки «Refresh», що надає можливість отримати останню інформацію про стан черги з серверу;
- кнопки «Back», що дозволяє закрити дане вікно та повернутися до головного меню;

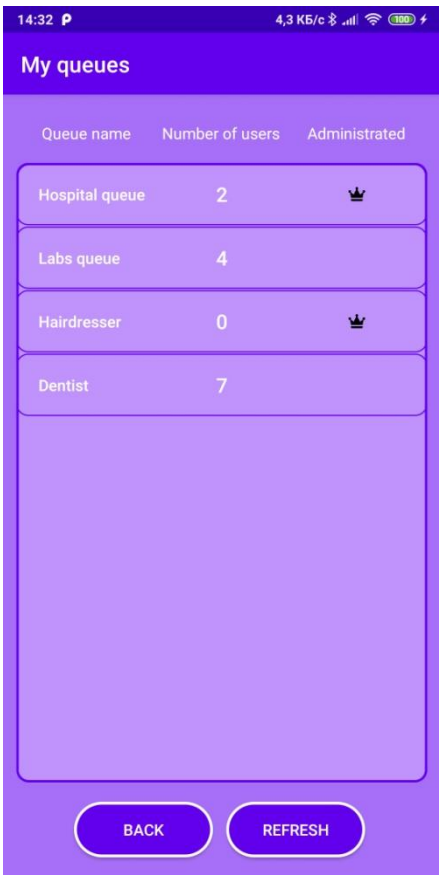

Рисунок 5.15 – Вікно черг користувача

Дане вікно відповідає за виведення активних черг користувача та надає можливість переходу до обраної черги по натиску на відповідний запис списку. Ця функція є зручною заміною переходу на сторінку адміністратора або користувача черги за допомогою кнопки «Connect queue» у головному меню.

Підводячи підсумок, розроблений в даному проєкті кінцевий продукт, що є системою формування та керування електронної черги для мобільних додатків, є повністю працеспроможним та задовольняє всі вимоги, що були поставлені до нього в технічному завданні проєкту.

ВИСНОВОК

При реалізації даного проєкту було розібрано багато компонентів. Насамперед, це система керування електронними чергами. Впродовж вивчення літератури було розібрано дане поняття та види даних систем. Також, детально розглядалися існуючі аналоги для розуміння вимог до продукту такого типу та окреслення особливостей власного проєкту.

Після визначення структури проєкту було обрано інструменти для її реалізації. Перш за все, відбулось детальне знайомство з операційною системою Android та розробкою програмного забезпечення на її базі. Було розібрано структуру додатку та нюанси його реалізації для даної платформи.

По завершенню вивчення реалізації додатку було перейдено до серверної частини. В процесі вивчення літератури було розібрано такі інструменти як Flask, REST, PostgreSQL, JSON та загальну структуру веб-додатку.

Після вивчення літератури було розроблено практичну частину, що являлася збіркою всього вище описаного та реалізовувала систему формування та контролю електронних черг для мобільних пристроїв.

Результатом роботи над проєктом є кінцевий продукт, що являє собою мобільний додаток для платформи Android з діючою серверною частиною. Додаток надає користувачу зручний та інтуїтивно зрозумілий інтерфейс для проведення маніпуляцій над електронними чергами та їх використання. Додаток проводить обмін даними з сервером за допомогою даних в JSON форматі та мережевого REST-ful інтерфейсу. Сервер реалізує основну логіку роботи з чергами та зберігає її результати на локальній машині й у базі даних.

Підсумком роботи над проєктом є здобуті знання в розробці клієнт-серверного додатку та готовий кінцевий продукт, що являє собою систему формування та керування електронної черги для мобільних пристроїв та задовольняє всі вимоги, що були поставлені до нього в технічному завданні проєкту.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Wavetec - Електрон. дані (1 файл) – Режим доступу: https://www.wavetec.com/solutions/queue-management/ (дата звернення 22.03.2020) – Назва з екрану.
2. Qmatic - Електрон. дані (1 файл) – Режим доступу: https://www.qmatic.com/solutions/queue-management/ (дата звернення 27.03.2020)– Назва з екрану.
3. AurionPro - Електрон. дані (1 файл) – Режим доступу: https://ace.aurionpro.com/what-is-queue-management-system/ (дата звернення 28.03.2020) – Назва з екрану.
4. StatCounter - Електрон. дані (1 файл) – Режим доступу: https://gs.statcounter.com/os-market-share/mobile/worldwide (дата звернення 15.04.2020) – Назва з екрану.
5. Android - Електрон. дані (1 файл) – Режим доступу: https://www.android.com/ (дата звернення 15.04.2020) – Назва з екрану.
6. Android Developers - Електрон. дані (1 файл) – Режим доступу: https://developer.android.com/ (дата звернення 10.04.2020) – Назва з екрану.
7. Flask - Електрон. дані (1 файл) – Режим доступу: https://flask.palletsprojects.com/en/1.1.x/ (дата звернення 15.04.2020) – Назва з екрану.
8. REST Api tutorial - Електрон. дані (1 файл) – Режим доступу: https://www.restapitutorial.com/ (дата звернення 16.04.2020) – Назва з екрану.
9. PostgreSQL - Електрон. дані (1 файл) – Режим доступу: https://www.postgresql.org/ (дата звернення 17.04.2020) – Назва з екрану.